World Wide Web (2014) 17:59–83
DOI 10.1007/s11280-012-0191-3

Evaluating continuous top-*k* queries over document streams

Weixiong Rao · Lei Chen · Shudong Chen · Sasu Tarkoma

Received: 18 January 2012 / Revised: 10 October 2012 /
Accepted: 29 October 2012 / Published online: 25 November 2012

Abstract At the age of Web 2.0, Web content becomes live, and users would like to automatically receive content of interest. Popular RSS subscription approach cannot offer fine-grained filtering approach. In this paper, we propose a personalized subscription approach over the live Web content. The document is represented by pairs of terms and weights. Meanwhile, each user defines a top-*k* continuous query. Based on an aggregation function to measure the relevance between a document and a query, the user continuously receives the top-*k* most relevant documents inside a sliding window. The challenge of the above subscription approach is the high processing cost, especially when the number of queries is very large. Our basic idea is to share evaluation results among queries. Based on the defined covering

Part of this work was done when the first author is currently affilicated with the Department of Computer Science, University of Helsinki, Finland and visting at China R&D Center for Internet of Things, Wuxi, China.

W. Rao (✉) · L. Chen
Computer Science & Engineering Department,
Hong Kong University of Science and Technology,
Clear Water Bay, Kowloon, Hong Kong, China
e-mail: wxrao@cse.ust.hk

L. Chen
e-mail: leichen@cse.ust.hk

S. Chen
Institute of Microelectronics of Chinese, Academy of Sciences,
Beijing, China
e-mail: chenshudong@ciotc.org

S. Chen
China R&D Center for Internet of Things, Wuxi, China

S. Tarkoma
Department of Computer Science, University of Helsinki,
Helsinki, Finland
e-mail: sasu.tarkoma@cs.helsinki.fi

relationship of queries, we identify the relations of aggregation scores of such queries and develop a graph indexing structure (GIS) to maintain the queries. Next, based on the GIS, we propose a document evaluation algorithm to share query results among queries. After that, we re-use evaluation history documents, and design a document indexing structure (DIS) to maintain the history documents. Finally, we adopt a cost model-based approach to unify the approaches of using GIS and DIS. The experimental results show that our solution outperforms the previous works using the classic inverted list structure.

Keywords top-k query • information filtering • web document streams

1 Introduction

At the age of Web 2.0, Web content becomes live. An example of such live content is the blog posting every day. In July 2006, there were over 1.6 million blog postings every day; the number of blogs worldwide was reported as 50 million and is doubled every 6 months [19]. Given such live content (instead of static content), many users nowadays would like to automatically receive content of interest with the help RSS subscriptions. However, the RSS subscriptions only offer coarse content filtering mechanism, typically treated as a *topic*-based filtering mechanism, and cannot meet the requirement of personalized subscriptions. For example, subscribers have to receive *all articles* posted by an subscribed RSS URL (e.g., associated with the news channel of *NY Times*), no matter really interested or not.

To overcome the above coarse filtering mechanism, in this paper, we propose a personalized subscription approach. Based on this approach, the document is modeled by pairs of terms and weights. The weights indicate the importance of the associated terms [15, 17, 26]. Meanwhile, each user defines a personalized top-k continuous query, consisting of input terms. Based on an aggregation function to measure the relevance between a document and a query, the user continuously receives the top-k most relevant documents within a sliding window.

The challenge of the top-k query model above is the high processing cost, especially when the number of queries is large, e.g., millions of queries. Two recent works [7, 11] consider optimization problems to minimize the evaluation cost of continuous queries over data stream systems. However, the queries in [7, 11], together with publish/subscribe systems [5], consist of boolean-based *selective filtering predicates*. That is, if and only if a publication item satisfies *all* predicates in a query, the item is said to satisfy the query. Unfortunately, the techniques [7, 11] developed for the boolean-based query cannot work for our problem. Specifically, the studied top-k query model adopts an aggregation function, and any (document) term, if having a non-zero weight, could contribute to the aggregation score. In addition, the top-k query on data stream systems [4, 8] and the subspace-based skyline approach [23] assume that data items and queries are associated with several (and at most tens of) dimensions. For the studied Web documents and queries, the overall semantic space involves millions of dimensions(i.e., the total number of terms). Therefore, the solutions proposed in [4, 8, 23] cannot be applied to our problem.

In this paper, our general idea of reducing the processing cost is to *share evaluation results among queries*. Specifically, we first define the *covering* relationship to identify the relations of aggregation scores among queries. Next, we develop *covering trees* to connect queries associated with the covering relations. In particular, instead of maintaining a large number of covering trees, we merge all covering trees to form a *covering graph*. Such a graph-based indexing structure, called GIS, uses the minimal number of edges to connect all queries. The document evaluation algorithm then benefits from the GIS to share the evaluation results with less evaluation cost.

Beyond GIS, we re-use the evaluation history to accelerate the evaluation of a new incoming document. To this end, we build a document-based indexing structure (i.e., DIS) to index the history documents. Finally, we adopt a cost model to unify the approaches using GIS and DIS. To summarize, our contributions are as follow.

- We exploit the covering relationship of queries to share evaluation results. We prove that the problem to build the covering graph with the minimal space cost is NP-hard. The proposed algorithm achieves a constant approximation ratio compared with the optimal result.
- We propose the evaluation algorithms using GIS and DIS, respectively. By a cost model, we further develop a unified evaluation approach with the least evaluation cost.
- Using real trace data sets, we verify that the proposed solution outperforms two previous works including the classic SIFT solution [26] and recent COL solution [6].

The rest of this paper is organized as follows. Section 2 first reviews related works. Section 3 then gives the data model and states the proposed problem. Next, Section 4 defines the covering relationship. Section 5 reports the details of GIS, and Section 6 presents evaluation algorithms. After that, Section 7 reports the experimental results. Finally, Section 8 concludes this paper.

2 Related work

We investigate the related work on information filtering (IF), publish/subscribe (pub/sub), and data stream systems.

In the context of information retrieval (IR), IF removes redundant or unwanted information from an information stream using (semi-) automated or computerized methods prior to presenting them to users. The classic work SIFT [26] utilized the *inverted list* to index continuous queries, which consist of terms and predefined *thresholds*. InRoute [1] is another centralized online filtering system, with help of inference networks to decide whether or not a document matches a query. [17] adopted query conditions similar to [26] and focused on reducing the information dissemination cost. Differing from these works adopting static thresholds, this paper tackles continuous top-k queries. Moreover, as shown in our experiments, only the inverted list (for example used by [26]), without exploiting query relations, is not enough to achieve high efficiency when the number of queries is large.

In pub/sub systems, subscribers define subscription query conditions (comparable to continuous queries) to declare their personal interests. Typically, a *subscription* query consists of a *conjunction* of selective *filters* (i.e., predicates). If one specific filter is dissatisfied with a publication, then all queries containing such a predicate are dissatisfied with the publication.[1] Compared with these boolean-based selective filtering model, the studied continuous top-k query involves an aggregation function, and the techniques of pub/sub systems are inapplicable.

In addition, the content-based pub/sub [5, 14, 20] explored the relationships of selective predicates and built various data structures (e.g., a dynamical R-tree [20]) to index subscription queries. However, these works are based on a low dimensional data schema (such as < 20 dimensions). Moreover, these previous works utilize *data ranges or intervals* of filters to define the overlapping or disjoint relationships. These defined relationships are also inapplicable to the studied continuous top-k queries that consist of query terms and adopt the aggregation function.

In the context of data streams, recent works [7, 11] adopted the boolean-based query model, where a query consists of multiple filters. This model is similar to the one in the content-based pub/sub. Both works focused on the optimal ordering problem to minimize the evaluation cost, and proposed a greedy strategy to first pick those filters associated with lower costs and then participate more queries. In addition, [4] tackled the ad-hoc query over low dimensional data items. The proposed techniques in [4] cannot be applied to our problem which is oriented for unstructured documents having millions of keywords in the whole semantic space. The subspace-based skyline approach [23] still considers the full dimensional space, which is similarly inefficient for our studied problem involving millions of terms.

Next, similar to our studied problem, [9] and its extension [10] monitor a stream of incoming documents for a set of users, who register their interests in the form of continuous top-k queries within a sliding window. Compared with our work, there exist significant differences as listed below. The Incremental Threshold Algorithm (ITA) proposed by [9] and [10] is based on the traditional inverted lists of queries; however, our solution leverages the developed covering trees and covering graphs. Moreover, when every document comes, ITA needs nontrivial efforts to incrementally maintain thresholds for the indexed queries. In our solution, the edges in the developed covering trees and covering graphs directly indicate the relations of the thresholds associated with endpoints of the edges. Thus we avoid the overhead to maintain thresholds as ITA did, and have chance to save efforts during the query processing.

After that, the recent work [6] overcomes the performance issue of [9] and [10] caused by profile indexing and result maintenance. By maintaining inverted lists of profiles, the main idea of [6] is to enable early stopping during the top-k processing and then to avoid processing all profiles in the inverted lists. The Completely Ordered Lists (COL)-based profile selection needs nontrivial cost to maintain the completely sorted lists all the time. For improvement, the Partially Ordered Lists (POL)-based profile selection does not keep complete order and maintains entries ordered only with regard to a number of boundaries. Differing from [6], we even avoid processing the whole inverted lists of some terms and our experiment shows that the proposed solutions use less cost than [6] does.

[1] Disjunction of selective filters can be intuitively treated as the union of multiple queries. Then each of those queries is still applicable for the claim that all filters are evaluated to be true.

Finally, our previous works [12, 13, 16, 18] designed solutions for distributed settings (such as DHTs [12, 13] and clusters of commodity machines [16]) to reduce the network traffic and parallel throughput. This paper instead designs solutions for a main-memory setting.

3 Preliminaries

In this section, we first introduce the data model (Section 3.1), define the problem (Section 3.2), and finally give the solution framework (Section 3.3). Figure 1 summarizes the main symbols and associated meanings in the paper.

3.1 Data model

We consider the model of the append-only document stream. That is, fresh documents are continuously appended to the end of the stream (e.g., RSS aggregators periodically download fresh articles posted in RSS feeds). After preprocessing, each document d is associated with $|d|$ pairs of $\langle t_j, s(t_j, d)\rangle$, where $|d|$ is the number of distinct document terms (or features) in d. The term score, denoted by $s(t_j, d)$, represents the importance or weight of t_j in d. Typically, $s(t_j, d)$ is pre-computed by the *term frequency* * *inverse document frequency* $(tf * idf)$ scheme, frequently used in IR domain. For other content like videos and photos, we can similarly assign the scores or weights to attributes or features pertaining to such content. Next, we normalize the term score by $ns(t_j, d) = s(t_j, d)/\sqrt{\sum_{j=1}^{|d|} s^2(t_j, d)}$. Moreover, we assume each document d is associated with an *expiration time* $expt(d)$, because only those unexpired documents are meaningful to users.

End users subscribe to their interested content by issuing a long-running query q, which is composed by $|q|$ query terms t_j $(1 \le j \le |q|)$, and a top-k number, denoted by $k(q)$. Following the classic Vector Space Model (VSM) in the IR context, we adopt

Symbol	Meaning
q_i, t_j and d	a query, term, and document
$s(t_j, d)$, $ns(t_j, d)$	score of term t_j in d, normalized score of t_j in d
$k(q)$, K_j, K	top-k number defined by q, maximal top-k number among all queries containing t_j, and maximal top-k number among all queries
$S(d, q)$, $TS(q)$, $TE(q)$	relevance score between d and q, threshold of relevance score of q, and threshold of expiration time of q
$\mathcal{Q}$, $\mathcal{D}$, $\mathcal{D}_k(q)$	set of queries, set of documents, top-k documents of q inside sliding window.
$\mathcal{R}$, $\mathcal{G}$, $\mathcal{T}$	Covering tree, covering graph, and set of all distinct terms in $\mathcal{Q}$
$\mathcal{R}_j$, $\mathcal{Q}_j$	covering tree rooted at a root query referred by term t_j, the queries that use the documents containing t_j as their top-k results

Figure 1 Used symbols and the meanings.

the following aggregation function to measure the *relevance* between document d and query q:

$$S(d, q) = \sum_{j=1}^{|q|} ns(t_j, d) \tag{1}$$

In the equation above, if a query term $t_j \in q$ appears in d, then the term score $ns(t_j, d)$ contributes to the relevance score $\mathrm{S}(d, q)$. Otherwise, if t_j does not appear in d, we intuitively treat $ns(t_j, d) = 0$ with no contribution to $\mathrm{S}(d, q)$. For a given q, a larger $\mathrm{S}(d, q)$ indicates that the document d is more similar to q and has more chance to become a top-k result of q. If the query q is exactly same to the document d, we have $\mathrm{S}(d, q) = 1$. Instead, if q and d do not contain any common term, we have $\mathrm{S}(d, q) = 0$.

Given (1) to compute $\mathrm{S}(d, q)$, our task is to report the top-k most *relevant* documents pertaining to each query q among all valid documents (i.e., those un-expired documents) inside a *sliding window* w. Window w can be two versions: a *counter* window w contains the most recent documents, and *time*-based window w contains all documents that arrives within a fixed time period covering the most recent timestamps. For convenience, our implementation considers the counter-based window.

Suppose query q is currently associated with the top-k documents d_n $(1 \le n \le k)$ inside the sliding window w. We denote those documents by $\mathcal{D}_k(q)$. The documents inside $\mathcal{D}_k(q)$ are associated with sorted relevance scores $\mathrm{S}(d_1, q) \ge ... \ge \mathrm{S}(d_k, q)$. The minimal relevance score $\mathrm{S}(d_k, q)$ is treated as the *threshold of relevance scores* of query q, denoted by $TS(q)$. Meanwhile, d_n becomes expired after the expiration time $expt(d_n)$. Similar to $TS(q)$, we define a *threshold of the expiration time*, $TE(q)$, as the minimal one among all expiration times $expt(d_n)$ of the documents inside $\mathcal{D}_k(q)$.

Now, a query q is associated with two thresholds $TS(q)$ and $TE(q)$, and we have three states pertaining to an incoming document d:

– If $\mathrm{S}(d, q) > TS(q)$ holds, d becomes a new *top-k result* of q, and is inserted to $\mathcal{D}_k(q)$. Meanwhile, d_k is dropped from $\mathcal{D}_k(q)$. Also, $TS(q)$ and $TE(q)$ are updated, if necessary.
– If both $\mathrm{S}(d, q) \le TS(q)$ and $expt(d) > TE(q)$ hold, d is a *top-k candidate* of q (but not inserted to $\mathcal{D}_k(q)$);
– If both $\mathrm{S}(d, q) \le TS(q)$ and $expt(d) \le TE(q)$ hold, d will never become a top-k result of q;

We use Figure 2 to illustrate the data model above. For each document d, the horizontal length indicates its expiration period (the moment of its right endpoint is the expiration time $expt(d)$), and the vertical height indicates the relevance score

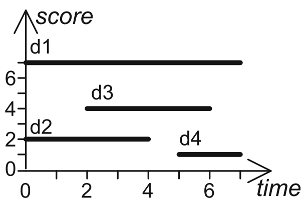

Figure 2 Continuous top-k query model with $k = 2$.

with a given query q. At time 0, for query q, $\mathcal{D}_k(q)$ contains documents d_1 and d_2 with $TS(q) = \mathrm{S}(d_2, q) = 2$ and $TE(q) = exp(d_2) = 4$. Next at time 2, due to $\mathrm{S}(d_3, q) > TS(q)$, the new document d_3 replaces d_2 and becomes a new top-k document. Meanwhile $TS(q)$ and $TE(q)$ are updated with $TS(q) = 4$ and $TE(q) = 6$. At time 5, because of $\mathrm{S}(d_4, q) \leq TS(q)$ and $expt(d_4) > TS(q)$, d_4 is kept as a top-k candidate, such that at time 6, d_3 is expired and d_4 becomes a new top-k result.

3.2 Problem statement

We state that the scheme to answer the continuous top-k query should meet the following requirements.

1. *Function Requirement*: the desirable scheme should correctly report the top-k documents of query q, and no top-k results are falsely missed.
2. *Efficiency Requirement*: the evaluation cost of the proposed scheme, measured by the evaluation time, should be low.
3. *Overhead Requirement*: the space cost to index queries is minimized. In this paper, we mainly consider the case that queries are maintained in main memory. Nevertheless, the proposed scheme can also be used for the other scenarios (e.g., queries are stored on disk-based file systems or databases).

Since both queries and documents in the proposed problem consist of terms, one possible approach is following the classic work [26] to index queries by an *inverted list*. That is, a *directory* maintains all distinct terms appearing in queries. Each term in the directory refers to a *posting list* which consists of all queries containing such a term. However, there is little chance for the inverted list to share evaluation results among queries, because the queries inside a posting list share only the associated term. Thus, the document evaluation algorithm has to scan *all* queries inside the posting list, incurring high evaluation cost. Instead, an ideal scheme evaluates a document d only with those queries which really need d as their top-k result, instead of all queries (because some of them do not need d).

On the other hand, to share evaluation results among queries, content-based pub/sub systems leverage *relations* of filters (comparable to queries in this paper), such as overlapping, disjoint, etc. to index subscriptions with various data structures [2, 20]. In this paper, based on such a general idea, we define the covering relations over queries to share evaluation results.

3.3 Solution framework

There are three main components in the solution framework. First, *query index*: we index all queries by a *graph-based indexing structure*, i.e., GIS. The GIS is built based on the defined *covering* relationship among queries. Those queries associated with *covering relations* can share the evaluation results. We present the covering relationship and the GIS in Sections 4 and 5, respectively. After that, for a new incoming document d, we use the GIS to evaluate d, and decide whether or not d is a top-k result/candidate of an indexed query q (Section 6.1).

Second, *document index*: beyond GIS, in Section 6.2, we further utilize the evaluation history to accelerate the evaluation, and index those already evaluated documents by a document indexing structure (DIS). Based on a cost model-based approach, we propose a unified solution to minimize the overall evaluation cost in Section 6.3.

Third, *maintenance*: when the document $d_e \in \mathcal{D}_k(q)$ is expired, we consider the maintenance of the current top-k documents and DIS. Then, before new documents come, we need to update $\mathcal{D}_k(q)$ and DIS to replace d_e with a valid top-k candidate d_c. We mainly follow [8], e.g., using the classic TA/NRA to find a candidate d_c with the top-1 largest relevance score pertaining to q.

In this paper, we mainly focus on the first two components. For the techniques of updating expired top-k documents inside $\mathcal{D}_k(q)$, please refer to previous works, e.g., [8].

4 Covering relationship

In this section, we develop the covering relationship of queries, based on which we further define the covering tree of queries.

4.1 Covering relation

Definition 1 (Covering relationship) If the query terms in q are the superset of the query terms in q', we define q *covers* q' (or q' is *covered* by q), denoted by $q \succeq q'$ or $q' \preceq q$.

Based on Definition 1, we have the following lemmas.

Lemma 1 *If $q \succeq q'$ and $q' \succeq q''$, then $q \succeq q''$.*

Lemma 2 *If $q \succeq q'$, then $\mathrm{S}(d, q) \geq \mathrm{S}(d, q')$ holds for any d.*

The lemmas above can be easily derived by following the definition of the covering relationship. Note that the definition and lemmas above can be easily extended to more complicated score functions (instead of (1)). For example, each $t_j \in q$ is associated with a preference weight $pw(t_j, q)$. We define $\mathrm{S}(d, q) = \sum_{j=1}^{|q|} [ns(dt_j, d) \cdot pw(t_j, q)]$; then $q \succeq q'$ holds if the condition $pw(t_j, q) \geq pw(t_j, q')$ is met for each $t_j \in \{q, q'\}$. After that, the lemmas still work.

Lemma 1 indicates the *transitive* property of the covering relationship, and will help define the covering tree. And Lemma 2 directly helps save the evaluation cost, which is shown as follows.

- Suppose q covers any $q' \in \mathcal{Q}$ where $\mathcal{Q}$ denotes a set of queries, and $TS_{lb}(\mathcal{Q})$ is the minimal $TS(q')$ for any $q' \in \mathcal{Q}$. If $\mathrm{S}(d, q) \leq TS_{lb}(\mathcal{Q})$ is satisfied, then for any $q' \in \mathcal{Q}$, the claim $\mathrm{S}(d, q') \leq TS(q')$ holds, without spending cost to evaluate d with q'.
- Alternatively, suppose q is covered by any $q' \in \mathcal{Q}$, and $TS_{ub}(\mathcal{Q})$ is the maximal $TS(q')$ for any $q' \in \mathcal{Q}$. If $\mathrm{S}(d, q) \geq TS_{ub}(\mathcal{Q})$ is satisfied, then for any $q' \in \mathcal{Q}$, the claim $\mathrm{S}(d, q') \geq TS(q')$ holds, without spending cost to evaluate d with q'.

Finally, in terms of $TS(q)$, we have:

Lemma 3 *Given $q \succeq q'$, if q and q' have the same top-k values, i.e., $k(q) = k(q')$, then $TS(q) \geq TS(q')$ holds.*

Proof The correctness of Lemma 3 is shown as follows. Recall that the definition of $TS(q)$ is the k-th largest one among all relevance scores between q and incoming documents. Since $k(q) = k(q')$, given $q \succeq q'$, we denote the documents having the k-th largest relevance score pertaining to q and q' by d_K and d'_K, respectively. Thus, $\mathrm{S}(d_K, q) \geq \mathrm{S}(d'_K, q')$ must be satisfied. Hence, Lemma 3 holds. □

4.2 Covering tree

Definition 2 (Covering tree) A covering tree $\mathcal{R}$ connects a set of queries, such that $q \succeq q'$ holds for any query q pointing to its non-null child query q'.

Following the definition, we have the following lemma.

Lemma 4 *Consider a query $q \in \mathcal{R}$ and any query q' inside its subtree rooted at q. For a document d, among all query terms $t_j \in q$ also appearing in d, $\mathrm{S}(d, q')$ is no larger than $\mathrm{S}(d, q)$, and no smaller than the minimal $s(t_j, d)$.*

Proof Following Definition 2, $q \succeq q'$ holds. This indicates that q' contains at least one query term of q. Otherwise, if none of query terms in q appear in q', the statement $q \succeq q'$ is invalid. Now because query q' contains at least one query term $dt_j \in q$, we have the lemma above. □

Lemma 4 helps find the lower and upper bounds of $\mathrm{S}(d, q')$, denoted by $\mathrm{S}_{\mathrm{lb}}(d, q')$ and $\mathrm{S}_{\mathrm{ub}}(d, q')$, respectively. Moreover, along the path from q to q' in $\mathcal{R}$, due to the transitive property, $\mathrm{S}_{\mathrm{lb}}(d, q')$ and $\mathrm{S}_{\mathrm{ub}}(d, q')$ become *tighter*, until $\mathrm{S}_{\mathrm{lb}}(d, q')$ and $\mathrm{S}_{\mathrm{ub}}(d, q')$ is finally equal to $\mathrm{S}(d, q')$. This property helps prune irrelevant documents during the traversal of $\mathcal{R}$ from the root to leaf nodes.

Next, we have the following lemma with respect to $TS(q)$.

Lemma 5 *For any $q \in \mathcal{R}$ and any q' inside its subtree in $\mathcal{R}$, if $k(q) \leq k(q')$ is met, then $TS(q) \geq TS(q')$ must hold.*

Following Lemma 5, we can find that for the root query $q \in \mathcal{R}$ and any other query $q' \in \mathcal{R}$, $TS(q)$ is the maximal one among all thresholds in $\mathcal{R}$. Also, given the root query q and a leaf query q', for all queries q'' inside the path from q to q', the associated score thresholds $TS(q'')$ are sorted in *descending* order.

Note that, Lemmas 3 and 5 require $k(q) \leq k(q')$. In case that each query defines its specific top-k number, Lemmas 3 and 5 may not hold. To ensure that Lemmas 3 and 5 work, a simple way is assuming each query defines the top-k number equal to K, where K is the maximal one among all $k(q)$ for any $q \in \mathcal{R}$. Then, among received top-K documents, each query further finds those really needed top-k documents. However, one issue is that each query receives too many useless documents, incurring high processing cost.

For optimization, we adopt the following tuning process. Suppose q is the parent of q' in $\mathcal{R}$; q and q' respectively define the top-k numbers equal to $k(q)$ and $k(q')$. If $k(q) \leq k(q')$, then the prerequisite of Lemma 5 holds; otherwise given $k(q) > k(q')$, we assume that q' also defines the top-$k(q)$ number, such that among top-$k(q)$ documents, q' can further find out those real top-$k(q')$ documents. By this approach, only those queries inside the subtree rooted at the query q defining a top-$k(q)$ number, will use the number $k(q)$ as their top-k numbers. This is helpful to overcome the issue of the above simple approach where all queries in $\mathcal{R}$ use the maximal number K as their top-k numbers.

5 Covering graph-based query index

In this section, we first give an overview of the covering graph in Section 5.1, and then present the details to build the graph in Section 5.2.

5.1 Overview of query index

The graph-based indexing structure (in short, GIS) contains a directed graph $\mathcal{G}$, and a directory of query terms. In $\mathcal{G}$, a vertex represents a query, and a directed link indicates that the source query *covers* the destination query. For query q, if the *traversal* of $\mathcal{G}$ starts from q, then the queries inside all traversal paths are covered by q. Moreover, those queries covered by q are connected in the form of a covering tree, which is rooted at q.

The motivation of maintaining a covering graph $\mathcal{G}$, instead of explicitly maintaining covering trees, is shown as follows. For a query q, it is possible for q to appear in multiple covering trees. If we explicitly maintain such covering trees, the query q (or the entry of q) is then maintained by multiple duplicate vertices, incurring *redundant space cost*. Due to a large number (e.g., million) of queries, explicitly maintaining covering trees obviously incurs *non-trivial space cost*. Instead, a covering graph $\mathcal{G}$ can index all queries, such that query q is now maintained as a single vertex.

Besides less space cost, the graph $\mathcal{G}$ can reduce the *evaluation cost*. It is because the proposed algorithm minimizes the number of edges of the graph $\mathcal{G}$, much smaller the number of edges maintained by the approach that explicitly maintains covering trees. Thus, the evaluation of a document d over the graph $\mathcal{G}$ uses less traversal cost, leading to a low evaluation time.

Finally, similar to an inverted list, GIS also maintains a *directory*. The terms in the directory refer to root queries. Specifically, for a term t_j, a referred root query must contain t_j. With the help of the directory, we find all those queries containing t_j by the traversal of $\mathcal{G}$ starting from those root queries referred by t_j, without the traversal of the whole graph $\mathcal{G}$. Thus, the directory helps save evaluation cost.

5.2 Details of covering graph

We first formally state the problem to build the covering graph $\mathcal{G}$ in Section 5.2.1, and then develop a greedy algorithm in Section 5.2.2.

5.2.1 Problem statement

In this section, we formally define the covering graph $\mathcal{G}$, which connects queries associated with covering relations, and meets two requirements: (i) all queries are connected to $\mathcal{G}$ with a minimal number of edges, and (ii) all queries containing any query term t_j are connected together to form covering graphs. Formally, given a set of queries $\mathcal{Q}$, we denote $\mathcal{T}$ to be the set of all distinct terms in $\mathcal{Q}$. For a term $t_j \in \mathcal{T}$ $(1 \leq j \leq |\mathcal{T}|)$ and a query $q_i \in \mathcal{Q}$ $(1 \leq i \leq |\mathcal{Q}|)$, if t_j appears in q_i, we define $I(i, j) = 1$; otherwise, $I(i, j) = 0$. For all such i and j, we denote the set of all $I(\cdot)$ by $\mathcal{I}$. We want to build a set of edges $\mathcal{E}$ to connect the queries in $\mathcal{Q}$ by the following problem.

Problem 1 (MIN_EDG) Given three sets $\mathcal{Q}$, $\mathcal{T}$, and $\mathcal{I}$, we build the set $\mathcal{E}$ to satisfy the two requirements above.

Theorem 1 *MIN_EDG is NP-Hard.*

Proof We prove the theorem by showing that the classic set-cover problem is a special case of MIN_EDG. Consider an instance of set-cover decision problem $(\mathcal{U}, \mathcal{S}, c)$ which consists of a universe set $\mathcal{U}$ and a collection of $\mathcal{S}$ of subsets of $\mathcal{U}$. The set-cover problem is to determine whether or not there is a set cover $\mathcal{C} \subseteq \mathcal{S}$ of size $|\mathcal{C}| = c$ such that $\bigsqcup_{s \in \mathcal{C}} = \mathcal{U}$, which is used to construct an instance of MIN_EDG as follows. We create $|\mathcal{T}|(= |\mathcal{U}|)$ terms that correspond to the elements of $\mathcal{U}$ in an one-to-one manner. Also, we create $|\mathcal{Q}|(= |\mathcal{S}|)$ queries w.r.t the elements of $\mathcal{S}$. For each query q_i with $1 \leq i \leq |\mathcal{Q}|$, we define its terms based on the elements of the corresponding subset $s \in \mathcal{S}$: q_i contains term t_j, if and only if s includes an element $u \in \mathcal{U}$ that corresponds to t_j. Finally, we take $|\mathcal{E}| = r$.

Consider a covering graph $\mathcal{G}$, such that for any $t_j \in \mathcal{T}$, all queries containing t_j are connected together. Denote $\mathcal{N}(q_i)$ to be the neighbors of q_i in $\mathcal{G}$: $\mathcal{N}(q_i=\{q_l|e(i, l) = 1$, with $1 \leq l \leq |\mathcal{Q}|\}$. Consider $\mathcal{C} = \{s|s \in \mathcal{S}$ corresponds to $q_l \in \mathcal{N}(q_i)\}$. Now, we will prove that there exists a query index related to this instance of MIN_EDG if and only if there exists a set cover related to the instance of the set cover problem by considering two following cases.

First, suppose there is a set cover with $|\mathcal{C}| = c$. We consider a covering graph $\mathcal{G}$ where all queries except a specific one q_i are connected with each other, and q_i is only connected with $\mathcal{N}(q_i)$ which corresponds to $\mathcal{C}$. Clearly, for every term t_j, there is a query $q_l \in \mathcal{N}(q_i)$ satisfying $I(i, l) = 1$, because otherwise $\bigsqcup_{s \in \mathcal{C}} = \mathcal{U}$ is invalid. Hence, all queries containing t_j are connected.

Second, as the reverse case, suppose there is a covering graph $\mathcal{G}$ satisfying (i) $|\mathcal{E}| \leq g$, and (ii) for any $t_j \in \mathcal{T}$, all queries containing t_j are connected. Due to $|\mathcal{E}| \leq g$, there must exist one specific query q_i satisfying $|\mathcal{N}(q_i)| \leq d$. Consider $\mathcal{C} = \{s|s \in \mathcal{S}, s$ corresponds to $q_l \in \mathcal{N}(q_i)\}$. Recall that $\forall t_j \in \mathcal{T}$, there must exist $q_l \neq q_i \in \mathcal{Q}$ satisfying $I(i, l) = 1$, because $\bigsqcup_{s \in \mathcal{C}} = \mathcal{U}$ is valid. Thus, $\forall t_j \in \mathcal{T}$, there must exist $q_l \in \mathcal{N}(q_i)$ satisfying $I(i, l) = 1$ because otherwise q_i is disconnected from those other queries containing t_j. Hence, $\bigsqcup_{s \in \mathcal{C}} = \mathcal{U}$ is satisfied, and $\mathcal{C}$ is the set cover. And $|\mathcal{C}| = |\mathcal{N}(q_i)| \leq c = g$ is satisfied. □

5.2.2 Greedy algorithm

Since MIN_EDG is NP-hard, we develop a greedy algorithm to build the covering graph $\mathcal{G}$, achieving a constant approximation ratio $O(|q|_{\max})$ compared with the optimal solution, where $|q|_{\max}$ is the maximal number of terms per query.

The intuition of the proposed greedy algorithm is as follows. In Figure 3b, query q_2 covers (or points to) 4 queries (q_1, q_3, q_4 and q_5), and q_6 covers 3 queries (q_3, q_4 and q_5). Based on the proposed greedy algorithm, in Figure 3c, q_2 points to only two queries (q_1 and q_4), and q_6 points to only one query q_4. Connecting two queries ensures that the source query covers the destination query with the largest *covering weight*, i.e., the number of query terms inside both the source and destination queries (we will show the meaning of the covering weight very soon). Also, a pair of connected queries are associated with a covering weight at least equal to 1. Meanwhile, all queries must be *resolved*. A resolved query q means that for each $t_j \in q$, q appears inside at least one covering tree with its root query referred by t_j, such that the traversal starting from the root queries referred by t_j can find q. Since each time the greedy algorithm adds query q to a covering tree by the largest covering weight, there are potentials to add q into all covering trees that q should appear in (due to the second requirement) with the smallest number of edges (due to the first requirement).

We define the covering weight as follows. For any two queries q_i and q_l associated with the covering relationship (we assume q_i covers q_l), we define the covering weight $w(i, l)$ to be the number of terms inside both q_i and q_l. The weight $w(i, l)$ indicates how fast a pair of queries is added to $\mathcal{G}$. Note that, if q_i and q_l have the exactly same terms, we can treated both as a single virtual query. It makes sense because the same terms of q_i and q_l indicate that they have the same relevance scores for any document d. Then, computing the relevance score between d and either of them correctly finds the relevance for both queries.

The general idea of Algorithm 1 is as follows. We first consider all potential edges to connect any two queries q_i and q_l associated with the covering relation having a weight $w(i, l) \geq 1$. Among such potential edges, we select a minimal number of edges to all vertexes (i.e., all queries). In this algorithm, we use a heap structure to maintain the all potential edges and the weights. For every t_j in each $q_i \in \mathcal{Q}$, we use a set G_{ij} to incrementally track the queries that will be added to the same covering tree as the query q_i. For a specific query q_i containing $|q_i|$ terms, we create the number $|q_i|$ of such sets. For the all queries in $\mathcal{Q}$, we create totally the number $\sum_{i=1}^{|\mathcal{Q}|} |q_i|$ of such sets.

Now, let us show the details of Algorithm 1. First, line 1 initiates the covering graph $\mathcal{G}$ by adding all queries as vertices of $\mathcal{G}$. Then, lines 2–3 initiate an empty subset G_{ij} for every term t_j appearing in each query $q_i \in \mathcal{Q}$ (we will gradually add more

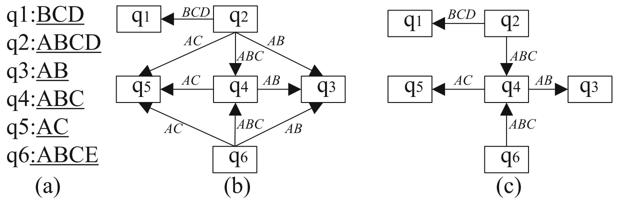

Figure 3 **a** 6 queries; **b** covering graph with all potential edges; **c** covering graph with minimal number of edges.

Algorithm 1: MIN_EDG (Queries $\mathcal{Q}$: $q_1 ... q_{|\mathcal{Q}|}$)

```
1  initiate covering graph $\mathcal{G}$ by adding all queries $q_i \in \mathcal{Q}$ as vertices;
2  for {each term $t_j$ appears in any $q_i \in \mathcal{Q}$} do
3      initiate an empty set $G_{ij}$;
4  for {any $1 \leq i < l \leq |\mathcal{Q}|$} do
5      do add pair $\langle i, l \rangle$ having weight $w(i, l) \geq 1$ to heap $\mathcal{H}$;
6  while {there still exists an unresolved query and $\mathcal{H}$ is not empty} do
7      pick the head pair $\langle i, l \rangle$ from $\mathcal{H}$;
8      create edge $q_i \rightarrow q_l$ if $q_i \succeq q_l$; otherwise add edge $q_l \rightarrow q_i$;
9      for {each $t_j \in q_i \cap q_l$} do
10         add the query $q_i$ to $G_{ij}$ (and add $q_l$ to $G_{lj}$), if not inside;
11         for {any $q_{i'} \in G_{ij}$, $q_{l'} \in \mathcal{G}_{lj}$ with $w(i', l') > 0$} do
12             update the weight $w(i', l')$ in $\mathcal{H}$;
13         for {each $q_{i'} \in G_{ij}$ (resp. $q_{l'} \in G_{lj}$)} do
14             $\mathcal{G}_{i'j} = G_{ij} \cup \mathcal{G}_{lj}$(resp. $\mathcal{G}_{l'j} = G_{ij} \cup \mathcal{G}_{lj}$);
15     if {$q_i$ is inside $G_{ij}$ (resp. $G_{lj}$) for each $t_j \in q_i$ (resp. $t_j \in q_l$)} then
16         resolve $q_i$ (resp. $q_l$);
17 for {each term $t_j$ appears in $\mathcal{Q}$} do
18     let term $t_j$ of the directory refer to root queries containing $t_j$;
```

queries to G_{ij} via the while loop and all queries inside G_{ij} are finally connected as a covering tree). Next, lines 4–5 first add all potential edges to the maximal heap $\mathcal{H}$ as soon as their associated weights are larger than zero.

After the initialization above, the while loop (lines 6–16) ensures that every query q_i is resolved or all items in $\mathcal{H}$ are processed. A *resolved* query q_i means that for all terms $t_j \in q_i$, q_i is inside the associated set G_{ij}, i.e., q_i should be inside the number $|q_i|$ of the sets G_{ij}. Inside the while loop, each iteration fetches from $\mathcal{H}$ a head pair $\langle i, l \rangle$, which is associated with the current largest weight $w(i, l)$. Line 8 then creates a directed edge $q_i \rightarrow q_l$ if q_i covers q_l (otherwise $q_l \rightarrow q_i$ if q_l covers q_i). Next for each term t_j commonly appearing in both q_i and q_l, we add the query q_i into the set G_{ij} w.r.t the query q_i and the term t_j, if q_i is not inside the set G_{ij} (resp. add q_l into the set G_{lj} w.r.t the query q_l and the term t_j if q_l not inside the set G_{lj}).

After that, we need to consider the member queries $q_{i'} \in G_{ij}$ and $q_{l'} \in G_{lj}$ if the associated weight $w(i', l') > 0$. If the new directed edge line 8 leads to a directed cycle between $q_{i'}$ and $q_{l'}$, i.e., $q_{i'} \succeq q_{l'}$ (or $q_{l'} \succeq q_{i'}$), lines 11–12 update the weight $w(i', l')$, if the pair $\langle i, l \rangle$ (plus the associated weight $w(i', l')$) is still inside the heap $\mathcal{H}$. The update leads to the original weight minus one by removing the effect of the common term t_j in line 9. This update makes sense because the new edge $q_i \rightarrow q_l$ in line 8 ensures that $q_{l'}$ is reachable from $q_{i'}$ (or $q_{i'}$ is reachable from $q_{l'}$). If the new value of the weight $w(i', l')$ becomes zero, this pair $\langle i, l \rangle$ will be removed from the heap $\mathcal{H}$.

In lines 13–14, for the query $q_{i'} \in G_{ij}$ and the term t_j, there exists a corresponding set $G_{i'j}$ (resp. for the query $q_{l'} \in \mathcal{G}_{lj}$, we have a set $G_{l'j}$). This is because lines 2–3 create a set G_{ij} for every $t_j \in q_i$. Then we need to update the set $G_{i'j}$ by the merged result of G_{ij} and $\mathcal{G}_{lj}$. After that, in line 15, if the resolving condition of q_i holds, then q_i is marked by the resolved state (similar situation occurs for q_l).

Finally, lines 17–18 focus on the post-processing task. For each query term $t_j \in \mathcal{Q}$, the term t_j maintained by the directory of GIS refers to all root queries containing t_j.

Running example Let us show a running example of Algorithm 1 as follows. Figure 3b shows the graph with all potential edges (i.e., all pairs of queries associated with the covering relation are connected), where the label of each edge indicates common terms of two endpoint queries. First, the heap $\mathcal{H}$ maintains 9 candidate edges that are shown in Figure 3b. Each subset G_{ij} is initiated with an empty set. When the pair $\langle 2, 1\rangle$ with the highest weight $w(2, 1) = 3$ is first fetched from $\mathcal{H}$, the edge $q_2 \rightarrow q_1$ is added to $\mathcal{G}$.

Next, during the **for** loop (lines 9–14), for each of the three terms $q_1 \cap q_2 = \{B, C, D\}$ (say term B), we make the following process. First, the query q_1 is added into the set G_{1B} w.r.t the query q_1 and the term B (and q_2 is to G_{2B} w.r.t the query q_2 and the term B). After that, lines 11–12, for the query $q_{i'} \in G_{ij}$ and $q_{l'} \in G_{lj}$ having the weight $w(i', l') > 0$, need to update the covering weight. Given $t_j = B, q_i = q_1$ and $q_{i'} = q_2$, we have $G_{1B} = \{q_1\}$, $G_{2B} = \{q_2\}$. Because the weight $w(1, 2) = 3$, we follow line 13 to update weight by $w(1, 2) = 3 - 1 = 2$. Next, line 14 updates the sets $G_{1B} = \{q_1, q_2\}$, $G_{2B} = \{q_1, q_2\}$. After running the two other terms $\{C, D\}$ in line 9, we have the weight $w(1, 2) = 0$, $G_{1C} = \{q_1, q_2\}$, $G_{2C} = \{q_1, q_2\}$, $G_{1D} = \{q_1, q_2\}$, and $G_{2D} = \{q_1, q_2\}$. Up to now, q_1 is resolved.

Next, $\langle 2, 4\rangle$ is fetched from $\mathcal{H}$, and the edge $q_2 \rightarrow q_4$ is added to $\mathcal{G}$. We update the weight $w(2, 4)$ and all subsets involving q_2 and q_4; both q_2 and q_4 are resolved. After that, similar situation occurs when $\langle 6, 4\rangle$ is fetched from $\mathcal{H}$.

Since the three queries (q_3, q_5 and q_6) are still unresolved and $\mathcal{H}$ is not empty, we need to fetch more pairs from $\mathcal{H}$. First, suppose $\langle 4, 5\rangle$ is fetched from $\mathcal{H}$, and the edge $q_4 \rightarrow q_5$ is added to $\mathcal{G}$. We update the subsets G_{5A} and G_{5C} and the weights $w(4, 5)$. Then, q_5 is resolved. Next, $\langle 2, 5\rangle$ is fetched from $\mathcal{H}$. However, the edge $q_2 \rightarrow q_5$ will not be added to $\mathcal{G}$, because the if condition of line 8 is not met. That is, for terms A and C, the query q_2 has already been added to G_{2A} and G_{2C}, and q_5 has already added to G_{5A} and G_{5C}. Similarly, the edge $q_4 \rightarrow q_3$ is added to $\mathcal{G}$, but the edges $q_2 \rightarrow q_3$, $q_6 \rightarrow q_5$, and $q_6 \rightarrow q_3$ are not. Thus, we get the covering graph $\mathcal{G}$ of Figure 3c.

Theorem 2 *The running time of Algorithm* 1 *is* $O((|\mathcal{Q}| \cdot |\overline{q}| - |\mathcal{T}|) \times \log(|\mathcal{Q}| \times |\overline{q}|))$, *where* $|\mathcal{T}|$ *and* $|\mathcal{Q}|$ *are the total number of distinct terms and queries, respectively;* $|\overline{q}|$ *is the average number of terms per query.*

Proof We assume that for a given term t_j, the number of queries containing containing t_j is Q_j. In terms of each root query containing term t_j, Algorithm 1 connects all those queries covered by such a root query as *a covering tree without cycles*, and connects all queries containing t_j as a *forest without cycles*. This is because line 8 ensures that q_i (or q_l) cannot be reachable from q_l (or q_i), and thus avoids cycles.

For each query term t_j, the forest pertaining to t_j contains Q_j queries and $(Q_j - 1)$ edges. Meanwhile, each iteration of the while loop adds only one edge. Thus, $(Q_j - 1)$ iterations are required to ensure that Q_j queries are connected to the forest. Since the total number of distinct terms is $|\mathcal{T}|$, the total number of iterations, which connect $|\mathcal{Q}|$ queries to at most $|\mathcal{T}|$ forests, is $\sum_{j=1}^{|\mathcal{T}|} (Q_j - 1)$. Here, $\sum_{j=1}^{|\mathcal{T}|} (Q_j - 1)$ can be further rewritten as $\sum_{j=1}^{|\mathcal{Q}|} |q_i| - |\mathcal{T}| = |\mathcal{Q}| \cdot |\overline{q}| - |\mathcal{T}|$, where $|q_i|$ is the number of terms in q_i, and $|\overline{q}|$ is the average number of terms per query.

Inside the while loop, the cost of each iteration is dominated by fetching the head item (line 10), and updating the weights of pairs in heap $\mathcal{H}$ (lines 14–15). Given the implementation of Fibonacci heap, the amortized cost of each iteration is $O(\log(|\mathcal{Q}| \times |\overline{q}|))$.

In addition, the initiation of the Fibonacci heap $\mathcal{H}$ and insertion of all none-zero pairs into $\mathcal{H}$ incur the amortized cost of $O(|\mathcal{Q}| \times |\overline{q}|)$.

Thus, the overall cost is $O((|\mathcal{Q}| \cdot |\overline{q}| - |\mathcal{T}|) \times \log(|\mathcal{Q}| \times |\overline{q}|))$. □

Theorem 3 *Compared with the optimal solution, Algorithm* 1 *has an approximation of at most* $|q|_{\max}$*, where* $|q|_{\max}$ *is the largest number of terms per query.*

Proof The proof is similar to the one of the approximation ratio of the set cover problem. First, the goal of MIN_EDG is to ensure that for each term t_j inside any query $q_i \in \mathcal{Q}$, q_i, together with other queries that also contain t_j, is added to a forest pertaining to t_j. It indicates that each query q_i is associated with at least one link to meet the goal of MIN_EDG; otherwise, q_i is disconnected from the forest. Next, each iteration can only connect only two queries by adding a single edge. This holds even for an optimal solution.

Now, in terms of any query q_i, we are interested in the number of iterations (and equally the number of edges) required to connect q_i with other queries. Recall that, Algorithm 1 connects q_i with another query (i.e., query q_l in line 10) per iteration. In the worst case, for each iteration, only one term commonly appears in both q_i and q_l. Thus, at most $|q_i|$ iterations (edges) are used to connect q_i with other queries.

Consider that $|q|_{\max}$ is the largest number of terms contained per query. For any query q_i, Algorithm 1 requires at most $|q|_{\max}$ folds of iterations (or edges) compared with the optimal solution. Hence, Algorithm 1 is valid. □

6 Document evaluation algorithm

Incoming documents are processed by the first-come and first-evaluate manner, such that users can receive fresh documents as early as possible. We first use GIS to evaluate the documents (Section 6.1). Next, we design a document-based indexing structure (DIS) to accelerate the document evaluation (Section 6.2), and develop a cost model-based approach to minimize the overall evaluation cost (Section 6.3).

6.1 Evaluating documents with GIS

Recall that only those terms appearing in both d and q contribute to $S(d, q)$. Thus, for each document term $t_j \in d$ $(1 \leq j \leq |d|)$, we use the directory of GIS to check whether or not there exist root queries containing t_j. If true, we retrieve the root queries referred by t_j, and evaluate d with those queries inside the covering tree rooted at such root queries (we denote such covering trees by $\mathcal{R}_j$). Otherwise, the evaluation with respect to t_j is unnecessary.

Since the term t_j might refer to root queries, there exist multiple associated covering trees, and all queries containing t_j must appear in such covering trees. To evaluate d with all queries containing t_j, a simple approach is to pick all the covering trees and then evaluate d with the queries inside all such trees. Instead, we notice that a query might appear in multiple covering trees. Thus, we propose an algorithm to choose only a subset of such covering trees and all queries containing t_j must appear in such chosen covering trees with no false dismissal. In this way, we can reduce the evaluation cost.

6.1.1 Evaluating queries inside covering trees

We first show how to evaluate d with queries inside a covering tree $\mathcal{R}_j$. Consider a query $q \in \mathcal{R}_j$ and any query q' inside the subtree rooted at q. By Lemmas 4 and 5, we have two rules to save the evaluation cost:

- **Rule 1**: If $\mathrm{S}_{lb}(d, q') \geq TS(q)$ is met, then for any query q' inside the subtree rooted at q, $\mathrm{S}(d, q') \geq TS(q')$ holds, indicating that d is the top-k document of q'.
- **Rule 2**: If $\mathrm{S}(d, q) < TS_{lb}(\mathcal{R}_j)$ is met, then for any query q' inside the subtree rooted at q, $\mathrm{S}(d, q') < TS(q')$ holds, indicating that d is not the top-k document of q'.

Given the rules above, the evaluation of d with all queries inside $\mathcal{R}_j$ can start from the associated root query by the breadth first algorithm to traverse the graph $\mathcal{G}$. During the traversal, if either of the above rules occurs, we directly determinate that d is (or not) the top-k result of all queries q' inside the subtree rooted at the current query q, and then terminate the traversal. Obviously, the rules above help reduce the evaluation cost.

6.1.2 Minimizing evaluation cost of GIS

Formally, we assume there exist $|d|'$ root queries and covering trees that are referred by all terms $t_j \in d$. Instead of choosing all such covering trees, we propose to optimally choose a subset of such trees to evaluate, meanwhile all queries containing the terms t_j are found with no false dismissals. We denote this problem to optimally choose covering trees by the MCT (i.e., short name of a $\underline{\text{m}}$inimal evaluation $\underline{\text{c}}$ost associated with chosen covering $\underline{\text{t}}$rees).

Theorem 4 *MCT is NP-hard.*

Proof The proof of this theorem can be easily reduced from the set-cover problem by treating common terms associated with any edge in $\mathcal{G}$ as a universal element U, and each query, consisting of those terms, as the subset $s \in S$. Then, MCT is equivalent to find the minimal number of evaluated queries $C \subseteq S$ to ensure that all edges, which can next find the connected queries, are found. □

Since MCT is NP-hard, we follow the set-cover greedy algorithm. The key of the set-cover greedy algorithm is the parameter c_j which measures the average evaluation cost per query in $\mathcal{R}_j$. Based on the set-cover greedy algorithm, we choose a covering tree $\mathcal{R}_j$ having the currently smallest c_j, and evaluate d with the queries inside the chosen $\mathcal{R}_j$. The process of choosing covering trees is repeated, until all queries containing any $t_j \in d$ are evaluated.

Now, we give the details to compute c_j as follows. Suppose that a covering tree $\mathcal{R}_j$ contains $|\mathcal{R}|_j$ queries. If no query in $\mathcal{R}_j$ needs d as their top-k document, then it is unnecessary to evaluate d with the queries in $\mathcal{R}_j$; otherwise, the evaluation is needed. Thus, for queries q_i $(1 \leq i \leq |\mathcal{R}|_j)$ in $\mathcal{R}_j$, we are interested in whether or not $\mathrm{S}(d, q_i) > TS(q_i)$ holds, even before d is really evaluated with q_i.

Given $|\mathcal{R}|_j$ queries in $\mathcal{R}_j$, we sum the all relevance scores with the document d, i.e.,$\sum_{i'=1}^{|\mathcal{R}|_j} \mathrm{S}(d, q_{i'})$, and the all thresholds, i.e., $\sum_{i'=1}^{|\mathcal{R}|_j} TS(q'_i)$. After that, if $\sum_{i'=1}^{|\mathcal{R}|_j} \mathrm{S}(d, q_{i'}) > \sum_{i'=1}^{|\mathcal{R}|_j} TS(q'_i)$ holds, there exists at least one query q_i satisfying $\mathrm{S}(d, q_i) > TS(q_i)$.

Until now, we measure the ratio c_j by $c_j = \frac{\sum_{i'=1}^{|\mathcal{R}|_j} \mathrm{S}(d,q_{i'})}{\sum_{i'=1}^{|\mathcal{R}|_j} TS(q'_i)}$, which is then helpful to determine the benefit of evaluating d with queries in $\mathcal{R}_j$ and whether d might become a top-k result of the queries in $\mathcal{R}_j$. Obviously, a larger c_j indicates more chance of those queries q_i satisfying $\mathrm{S}(d, q_i) > TS(q_i)$.

To compute c_j, we note that the denominator $\sum_{i'=1}^{|\mathcal{R}|_j} TS(q'_i)$ is irrelevant to the document d, and pre-compute it before the document evaluation. After that, the key to compute c_j is $\sum_{i'=1}^{|\mathcal{R}|_j} \mathrm{S}(d, q_{i'})$, which is transformed as follows.

$$\begin{aligned}
\sum_{i'=1}^{|\mathcal{R}|_j} \mathrm{S}(d, q_{i'}) = &\sum_{i'=1}^{|\mathcal{R}|_j} \left[\sum_{j'=1}^{|q_{i'}|} s(t_{j'}, d) \right] \\
&= \sum_{i'=1}^{|\mathcal{R}|_j} \left[\sum_{j'=1}^{|d|} s(t_{j'}, d) \cdot \theta_{i'j'} \right] \\
&= \sum_{j'=1}^{|d|} \left[\sum_{i'=1}^{|\mathcal{R}|_j} s(t_{j'}, d) \cdot \theta_{i'j'} \right] \\
&= \sum_{j'=1}^{|d|} \left[s(t_{j'}, d) \cdot \sum_{i'=1}^{|\mathcal{R}|_j} \theta_{i'j'} \right]
\end{aligned}$$

In the above transformation, we first define a binary coefficient $\theta_{i'j} = 1$ if the document d contains the the term $t_j \in q_{i'}$, and $\theta_{i'j} = 0$ otherwise. Based on the $\theta_{i'j}$, we then transform $\sum_{j=1}^{|q_{i'}|} s(t_j, d) = \sum_{j=1}^{|d|} s(t_j, d) \cdot \theta_{i'j}$ where $|d|$ is the number of terms in d.

Now we denote $|\mathcal{R}|_{jj'}$=$\sum_{i'=1}^{|\mathcal{R}|_j} \theta_{i'j'}$, the number of queries in $\mathcal{R}_j$ that contain a term $t_{j'} \in d$. The key observation of the above transformation is that we can easily precomputed the value of $|\mathcal{R}|_{jj'}$ before the document evaluation. Moreover, among those queries in $\mathcal{R}_j$, if more queries contain $t_{j'} \in d$, the value $|\mathcal{R}|_{ij'}/|\mathcal{R}|_j$ is larger. Intuitively, $|\mathcal{R}|_{ij'}/|\mathcal{R}|_j$ indicates the *popularity* of the term $t_{j'} \in d$ in $\mathcal{R}_j$.

Now, before evaluating d, we can summarize queries in $\mathcal{R}_j$ and precompute some statistical results, e.g., $|\mathcal{R}|_{ij'}$. When d comes, we then compute the sum $\sum_{i'=1}^{|\mathcal{R}|_j} \mathrm{S}(d, q_{i'})$ and c_j, even without having the details of these queries. Given the values c_j associated with all covering trees, we follow the set-cover greedy algorithm, and choose covering trees with the currently least c_j in order to minimize the overall document evaluation cost.

6.2 Evaluation with DIS

Differing from GIS, DIS re-uses previous evaluation results to accelerate the evaluation. The general idea of using DIS is shown as follows. With the help of DIS to index those already evaluated documents, we measure whether or not an incoming document d is similar to those indexed documents d'. If d' is a top-k result of query q, then d, if similar to d', has the potentials to become a top-k result of q, too.

Based on the idea above, we need to (i) index those already evaluated documents d', and (ii) maintain a set of queries which use d' as their top-k results by a vector.

First to index the already evaluated documents d', we utilize an *inverted list*. However, differing from the traditional approach, we do not add d' to the posting lists of all document terms $t_j \in d'$. Instead, we add d' to a small number of posting lists. That is, for a document term $t_j \in d'$, we denote K_j to be the maximal top-k number among all queries containing the term t_j. Next, we use the posting list of t_j to index the K_j documents d' having the largest $s(t_j, d')$ among all valid documents inside the sliding window. Finally, we denote $\mathcal{Q}_j$ to be the queries which use the indexed documents in the posting list of t_j as their top-k results, and index the queries by a vector.

Given the above DIS, we evaluate each incoming document d as follows. In order to find those indexed documents d' that are similar to d, each term $t_j \in d$, we determine whether or not DIS has indexed the documents d' in the posting list of t_j having the terms cores $s(t_j, d') > s(t_j, d)$. If true, we retrieve the queries in $\mathcal{Q}_j$, and evaluate d with each query in $\mathcal{Q}_j$.

We note that the maintenance of DIS is needed in case that new documents come and old document expired. For such maintenance, the previous work [8] provided the 2-dimensional skyline approach to cover both the time expiration and relevance score. In this case, once the expired documents are removed from the skyline, the indexed documents in DIS are then removed, too.

6.3 A unified solution

Intuitively, GIS ensures that *all* queries can receive their needed top-k results with no false dismissals, and DIS re-uses the previous results to accelerate the evaluation. In this section, we unify both approaches to evaluate documents with minimal evaluation cost and without false dismissals.[2]

We expect the unified solution spends less evaluation cost than the approach using only GIS. Suppose term $t_j \in d$ appears in both the directory of GIS and the inverted list of DIS. Recall that $\mathcal{R}_j$ denotes the covering trees in GIS referred by t_j, and $\mathcal{Q}_j$ denotes the queries that use the indexed documents in the posting lists of t_j as their top-k results. Suppose the cost of using GIS to evaluate queries in $\mathcal{R}_j$ is $Cost(\mathcal{R}_j)$, and the cost of using DIS to evaluate queries in $\mathcal{Q}_j$ is $Cost(\mathcal{Q}_j)$. Because the queries in $\mathcal{Q}_j$ are only the subset of those in $\mathcal{R}_j$ (i.e., $\mathcal{Q}_j \sqsubseteq \mathcal{R}_j$), the cost of the unified approach spends $Cost(\mathcal{Q}_j)$+$Cost(\mathcal{R}'_i)$, where $Cost(\mathcal{R}'_i)$ is the cost to evaluate the remaining queries in $\mathcal{R}_j$ except for those queries in $\mathcal{Q}_j$. To

[2]In the inverted list of DIS, $\mathcal{Q}_j$ does not contain all queries containing t_j. Thus, the approach of using DIS, though accelerating the evaluation with queries $q \in \mathcal{Q}_j$, cannot guarantee an incoming document d is evaluated with all queries that need d as their top-k result. It means d might be falsely missed for queries not inside $\mathcal{Q}_j$.

ensure the unified approach outperforms the pure approach using GIS, the condition $Cost(\mathcal{Q}_j) \leq Cost(\mathcal{R}_j) - Cost(\mathcal{R}'_i)$ must hold.

To ensure the condition $Cost(\mathcal{Q}_j) \leq Cost(\mathcal{R}_j) - Cost(\mathcal{R}'_i)$ holds, among the terms $t_j \in d$, we follow [17, 21] to select the ones having the top (e.g., $p = 10$ %) highest term scores, and evaluate documents with queries in $\mathcal{Q}_j$. Meanwhile, we measure the associated evaluation cost $Cost(\mathcal{Q}_j)$ (e.g., the consumed time). Next, we continue the evaluation of remaining queries in $\mathcal{R}_j$ (i.e., $\mathcal{R}'_i$), and measure its evaluation cost $Cost(\mathcal{R}'_i)$. Based on the cost $Cost(\mathcal{R}'_i)$ and the numbers of queries in $\mathcal{Q}_j$ and $\mathcal{R}'_i$, we approximate the cost $Cost(\mathcal{R}_j)$. If $Cost(\mathcal{Q}_j) \leq Cost(\mathcal{R}_j) - Cost(\mathcal{R}'_i)$ holds, then the unified approach outperforms the pure approach using only GIS. In our experiment, for $p = 10$ %, the experimental results verify that the unified approach outperforms the pure GIS approach by folds of faster running time.

7 Experiments

We first show the used data sets (Section 7.1). Next, we compare our solution with two counterparts in terms of the space cost and the evaluation cost (Sections 7.2 and 7.3).

We compare our approaches (i.e., the algorithm using only GIS and the unified approach) with COL in [6] and SIFT [26]. We implemented all approaches based on main memory-based indexes by Java 1.6. Documents are read from the local disk as a stream to evaluate with indexed queries. All experiments are tested in a Linux server with 4GB memory and Intel Xeon 3.00GHz CPU.

7.1 Data sets

We respectively describe the data sets used as queries and documents in Sections 7.1.1 and 7.1.2.

7.1.1 Query logs

Google Alerts and Microsoft Live Alerts provide input interfaces by which end users subscribe to favorable documents via input keywords. Unfortunately, there is no publicly available real data set about using keywords as continuous queries. We note that there are difference between the search queries and the continuous queries. Nevertheless, these search logs truly show behaviors of end users to use keywords. Moreover, using search query logs as continuous queries is frequently used in previous works [24, 25]. Thus, we use two real data sets of traditional search query logs as our continuous queries.

We use two real query logs: (i) a trace log with 81.3 MB collected within four months from a popular commercial search engine (in short `commercial SE`), which is quite representative for the behaviors of end users in the real world; (ii) a similar query log from www.search.com with size 1.27 MB (in short `search.com`). Table 1 summarizes the parameters of both query logs. On average, the number of terms per query is 2.085 in `search.com`, and 2.843 in the `commercial SE`; the largest number of terms per query in `search.com` is 11, and that number in the `commercial SE` is 29. Furthermore, Figure 4a plots the number of terms of two query logs, where the x-axis represents the number of terms, and the

Table 1 Statistics of two query logs.

Query parameter	Search.com	Commercial SE
Total num. of queries	81,497	4,000,000
Total num. of distinct terms	41,722	1,038,567
Avg. num. of terms per query	2.085	2.843
Max. num. of terms per query	11	29
Min. num. of terms per query	1	1

y-axis represents the percentage of queries with the corresponding number of terms. For example, for the `search.com` trace, the percentages of queries consisting of 1, 2, 3, and 4 terms are 38.13, 33.09, 17.56, and 7.05 %; for the `commercial SE` trace, the percentages of queries consisting of 1, 2, 3 and 4 terms are 31.33, 36.42, 17.56, and 7.39 %. For both data sets, the standard deviation values over the overall average query length are 1.908 and 2.043, respectively. Both deviation values indicate that most queries consist of around 2–3 terms. Clearly, Table 1 and Figure 4a indicate that input queries in both trace files are typically composed of very few terms. Hence, if these input queries are used as continuous queries, there are opportunities for our proposed approaches to save the evaluation cost.

Next, Figure 4b plots the statistics of the covering relations of the two traces, where x -axis is the number of terms per query (i.e., the query length $|q|$) and y-axis indicates the rate of the associated queries which are covered by other queries. As shown in this figure, the queries with small length have more chance to be covered. For example for $|q| = 1$, in the `Search.com` and `commercial SE`, 53.2 and 42.3 % queries are covered respectively, indicating that such queries are inside the covering graph to save evaluation cost. On the overall, around 42.5 and 31.4 % queries (including those root queries which are not covered by any other queries) are added to the covering graph. We note that the remaining queries are isolated from the covering graph. Fortunately, such queries are accessible with the help of the directory structure of GIS, which are referred by the terms in the quires. In this way, there is no significant degradation of the evaluation cost to access the isolated queries.

7.1.2 Documents

We use two data sets (Table 2 summarizes the statistics for the test data sets):

(1) one based on Text Retrieval Conference (TREC) WT10G web corpus, a large test set widely used in web retrieval research. The dataset contains around 10 gigabyte, 1.69 million web page documents and a set of queries (we mean

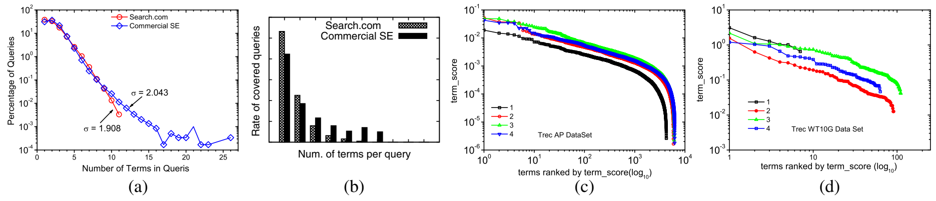

Figure 4 **a** Two query logs with short terms; **b** Statistics of covered queries; **c** TREC AP with large articles; **d** TREC WT10G with short articles.

Table 2 Statistics of the TREC data sets.

Document parameter	TREC WT10G	TREC AP
Total num. of docs	1,692,096	1,050
Avg. num. of terms per doc	64.808	6,054.9
Max. num. of terms per doc	331	7,320
Min. num. of terms per doc	2	1,303
Total num. of terms	16,382	48,788
Avg. term frequency	130.31	619.48
Max. term frequency	210,089	1,050
Min. term frequency	1	1

the "title" field of a TREC topic as a query). The WT10g data was divided into 11,680 collections based on document URLs. Each collection on average has 144 documents with the smallest one having only 5 documents. The average size of each document is 5.91KB. The data set was stemmed with the Porter algorithm and common stop words such as "the", "and", etc. were removed from the data set.

(2) one based on TREC AP: a text categorization task based on the Associated Press articles used in the NIST TREC evaluations. Compared with TREC WT10G data set, the TREC AP data set is composed of fewer (only 1,050) articles but with a larger number of terms, on average 6054.9 per article.

By formula $score(t_i, d) = \frac{freq_{i,d}}{Max_l(freq_{l,d})} \cdot \log \frac{N}{n_i}$, we compute the score of each term in both data sets. In this formula, $\frac{freq_{i,d}}{Max_l(freq_{l,d})}$ represents the value of term frequencies (tf), and $\log \frac{N}{n_i}$ the inverse document frequencies (idf). In details, $freq_{i,d}$ is the frequency of term t_j in document d, and $Max_l(freq_{l,d})$ is the maximal term frequency in d; n_i is the number of documents containing t_j across the whole data set, and N is the total number of documents.

Though our experiments pre-compute $score(t_i, d)$ by the approach above, it has shown that in IR scenarios it is enough to have an approximation of idf values [3]. This approximation is useful for our work in the scenario of a stream of documents. Though it is an open problem to approximate idf in such a scenario [24], one possible solution is to use the similar approximate solution as [22] to compute idf. After the value of idf is ready, we can then easily compute $score(t_i, d)$.

For illustration, we use Figs. 4c–d to plot the term scores of four randomly chosen documents from the TREC AP and WT10g data sets, respectively. Though the sampled documents might not be enough to represent the entire data set, they are helpful to give an intuition of the distribution of term scores in these documents. Clearly, the skewed distribution of term scores in both ïňĄgures is useful for our solution: if the queries containing the terms having high term scores, such queries have significantly larger relevance scores and thresholds than those queries containing the terms having low term scores. Thus, it makes sense to choose the document terms having the top term scores during the unified evaluation (see Section 6.3).

7.2 Space cost of query indices

For the `search.com` traces, we use entries of this trace as queries, and index them by the GIS and inverted list (both COL ad SIFT used the inverted list) respectively.

Besides query terms, we randomly generate the top-k number within $[1, K]$, where K is a given maximal top-k number.

In terms of the space cost of both indices, we mainly use two metrics: (i) the total number of edges, and (ii) the average number of edges per vertex, i.e., the average degree per vertex. For GIS, queries are connected by treating queries as vertices. Since the GIS is implemented by the adjacent list, the total number of edges directly indicates the space cost of the GIS. Note that those root queries referred by each query term, t_j, in the directory of GIS are implemented as a vector, for fairness we also consider those root queries use $(R_i - 1)$ edges, where R_i is the size of such vector. In the extreme case that none pair of queries are connected to form a covering tree, the covering graph is actually an inverted list, and R_i is just equal to the size the posting list pertaining to t_j. For the inverted list, the total size of all posting lists directly decides its space cost.

Figure 5a plots the total number of edges in the GIS and in the inverted list for the `search.com` trace. When the number of queries grows, the total numbers of edges of both approaches grow as well. However, since we propose to merge all covering trees into the covering graph, the total number of edges in GIS is smaller than that in the inverted list. For example, when the number of queries is 81479, the number of edges in the inverted list is 2.16 folds of that in the GIS.

Next, Figure 5b plots the average degree per vertex of the GIS and the average size per posting list for the `search.com` trace. In this figure, more queries lead to higher average degrees for both approaches. When the number of queries is 81479, for the inverted list, the average size per posting list is 3.29; for GIS, the average degree per vertex is only 0.79 (smaller than 1). It is because some queries donot have any covering relations and are isolated from the GIS (indicating the degree of such queries is 0).

For the `commercial SE` trace, we plot the total number of edges and the average degree per vertex in Figure 5c–d. This figure shares the similar trend as Figure 5a–b. For example, when the total number of queries is 2×10^6, the number of edges in the inverted list is 1.38 folds of that in the GIS, and the average degree per vertex in the inverted list is 3.01 folds of that in the GIS. These numbers help save the evaluation cost for the GIS and unified approach.

As a summary, GIS uses fewer edges to connect all queries than the inverted list does. The fewer edges and degrees help GIS spend less cost to traverse the graph and find indexed queries. Thus, GIS uses less cost to evaluate incoming documents with indexed queries. The following experiments in terms of the running time will verify the claim.

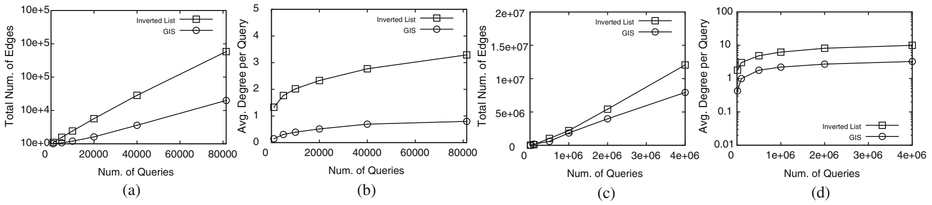

Figure 5 Index comparison. *Search.com* trace: (**a**) edges and (**b**) degree; Commercial SE trace: (**c**) edges and (**d**) degree.

7.3 Running time

To evaluate documents against indexed queries, we assume a counter-based window with size $W = 100$, and $|\mathcal{D}|$ documents arrive at the system, and each document is randomly assigned an expiration time within [0.0, 1.0]. For `Search.com` and TREC AP trace, Figure 6 studies the effects of three parameters: (i) the number of queries $|\mathcal{Q}|$, (ii) the number of incoming document $|\mathcal{D}|$, and (iii) the maximal top-k number K. We study how these parameters affect the running time, and compare our approach using GIS with the approach using the inverted list [26]. By default, $|\mathcal{Q}|$, $|\mathcal{D}|$, and K are 81479, 100, and 10, respectively.

In Figure 6a, when the number of queries grows from 1000 to 81479, the running time of all four approaches grows as well. That is, given more queries, both SIFT and COL have to check the more posting lists (due to more number of terms that commonly appear in the document and queries); the two GIS approaches also have to check more covering trees rooted at such common terms. Thus, more indexes lead to higher running time. In addition, the COL still uses the inverted list structure. Thus, a query Id could duplicately appear in multiple posting lists. Instead, the GIS avoids the redundancy and reduces the traversal cost during the evaluation, and the GIS approaches leverage covering trees to reduce the evaluation cost with less processing time than COL.

Figure 6b shows that a larger number of new documents, i.e., $|\mathcal{D}|$, can help reduce the average evaluation cost per document for the four approaches. It is because for a specific query q, more documents are published to evaluate with q, which leads to a larger $TS(q)$ (i.e., the top-k threshold of q). Since we fix the top-k number, more documents are pruned, resulting in lower average evaluation cost per document for all four approaches. In particular, we observe that the unified approach benefits more from a larger value of $|\mathcal{D}|$. For example, for $|\mathcal{D}| = 10$, the running time of the unified approach is 0.702 of the GIS approach. Instead for $|\mathcal{D}| = 1000$, the running time of the unified approach is around 0.3645 of the GIS approach.

Finally, Figure 6c shows the effect of the defined maximal top-k number. For both approaches, a larger top-K number incurs higher running time because a larger top-K number indicates lower thresholds $TS(q)$. For example, for the top-K numbers equal to 10, the running time per document used by the GIS approach is 22.32 ms; for the top-K number equal to 50, the running time is 63.64. Similar situation occurs for the other three approaches. Note that the unified approach benefits less from a larger top-k number. For example, for $K = 1$, the average running timer of the

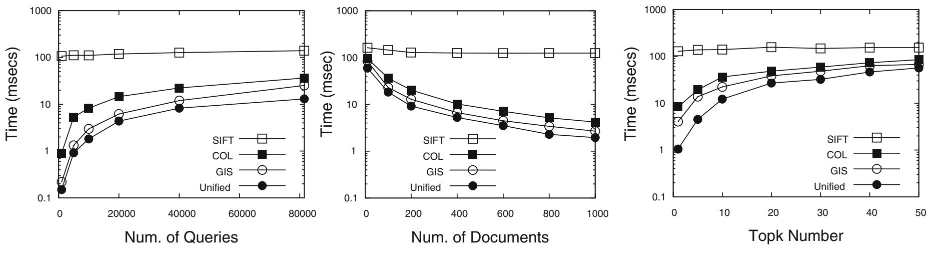

Figure 6 Running time for the search.com trace and TREC AP traces.

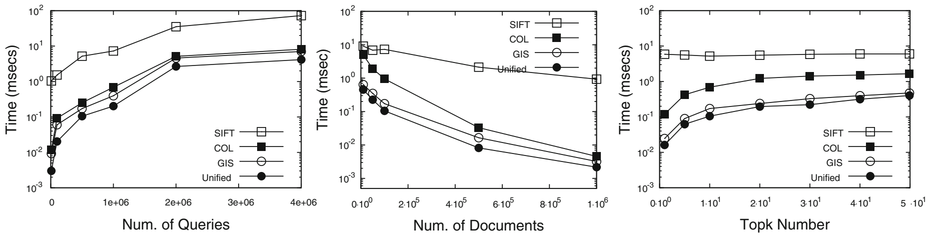

Figure 7 Running time of the commercial trace and TREC WT10g.

unified approach is only 24.5 % of the GIS approach. When K becomes larger, such as $K = 50$, the running time of the unified approach is 83.98 % of the GIS approach.

Next, we use `commercial SE` and TREC WT data sets and plot the associated running time in Figure 7. In Figure 7a, larger number of queries (from 1,000 to $4{*}10^6$) also leads to larger running time. This is consist with Figure 6a. However, given the same number of queries (e.g., 10,000), the running time of the GIS approach is only 0.0603 ms per document, significantly smaller than the used time (8.538 ms per document) in Figure 6a. It is because the average number of terms per document for TREC WT is smaller than that for TREC AP (refers to Figure 1).

As shown in Figure 7b, when more new documents arrive per timestamp, the running time of the four approach is reduced, which is similar to Figure 6b. However, due to significantly larger number of documents used in this experiment than Figure 6b, the average running time per document shown in this figure is much more smaller than Figure 6b.

Finally, in Figure 7c, a larger top-K number leads to higher running time of the approaches. Consistent with Figure 6c, the running time of SIFT and COL is larger than that of the pure GIS approach. For example, for the top-K number equal to 100, SIFT consumes 8.075 folds larger running time than the pure GIS approach.

8 Conclusion and future work

In this paper, we study the problem of continuous top-k queries over a stream of documents. Given both queries and documents consisting of terms, previous works heavily rely on the inverted list to index queries and evaluate documents. Instead, we define the covering relation, and index queries by the covering graph. The proposed document evaluation algorithm leverages the covering graph to share evaluation results among queries. Our experiments based on real data sets show the proposed solution can achieve much better evaluation results than the inverted list solutions SIFT and COL. As future work, we consider two directions: (i) more semantical data models (such as predicate-based query conditions) the corresponding solutions, and (ii) more scalable and distributed solutions (e.g., on a cluster of commodity machines). Finally, we note that there exist differences between search queries (used in our experiments) and filtering queries. For example, the number of terms in filtering queries is typically larger than the one in search queries. As an important future work, we plan to collect filtering queries from real applications and then evaluate the proposed algorithms based on such filtering queries.

References

1. Callan, J.P.: Document filtering with inference networks. In: SIGIR, pp. 262–269 (1996)
2. Chandramouli, B., Phillips, J., Yang, J.: Value-based notification conditions in large-scale publish/subscribe systems. In: VLDB, pp. 878–889 (2007)
3. Cuenca-Acuna, F.M., Nguyen, T.D.: Text-based content search and retrieval in ad-hoc p2p communities. In: Networking Workshops, pp. 220–234 (2002)
4. Das, G., Gunopulos, D., Koudas, N., Sarkas, N.: Ad-hoc top-k query answering for data streams. In: VLDB, pp. 183–194 (2007)
5. Fabret, F., Jacobsen, H.A., Llirbat, F., Pereira, J., Ross, K.A., Shasha, D.: Filtering algorithms and implementation for very fast publish/subscribe. In: SIGMOD Conference, pp. 115–126 (2001)
6. Haghani, P., Michel, S., Aberer, K.: The gist of everything new: personalized top-k processing over web 2.0 streams. In: CIKM, pp. 489–498 (2010)
7. Liu, Z., S.P. 0002, Ranganathan, A., Yang, H.: Near-optimal algorithms for shared filter evaluation in data stream systems. In: SIGMOD Conference, pp. 133–146 (2008)
8. Mouratidis, K., Bakiras, S., Papadias, D.: Continuous monitoring of top-k queries over sliding windows. In : SIGMOD Conference, pp. 635–646 (2006)
9. Mouratidis, k., Pang, h.: An incremental threshold method for continuous text search queries. In: ICDE, pp. 1187–1190 (2009)
10. Mouratidis, K., Pang, H.: Efficient evaluation of continuous text search queries. IEEE Trans. Knowl. Data Eng. **23**(10), 1469–1482 (2011)
11. Munagala, K., Srivastava, U., Widom, J.: Optimization of continuous queries with shared expensive filters. In: PODS, pp. 215–224 (2007)
12. Rao, W., Chen, L.: A distributed full-text top-k document dissemination system in distributed hash tables. World Wide Web **14**(5–6), 545–572 (2011)
13. Rao, W., Chen, L.: Distributed top-k full-text content dissemination. Distributed and Parallel Databases **30**(3–4), 273–301 (2012)
14. Rao, W., Chen, L., Fu, A.W.: On efficient content matching in distributed pub/sub systems. In: INFOCOM (2009)
15. Rao, W., Chen, L., Fu, A.W.C.: Stairs: towards efficient full-text filtering and dissemination in dht environments. VLDB J. **20**(6), 793–817 (2011)
16. Rao, W., Chen, L., Hui, P., Tarkoma, S.: Move: a large scale keyword-based content filtering and dissemination system. In: ICDCS, pp. 445–454 (2012)
17. Rao, W., Fu, A.W.C., Chen, L., Chen, H.: Stairs: towards efficient full-text filtering and dissemination in a dht environment. In: ICDE (2009)
18. Rao, W., Vitenberg, R., Tarkoma, S.: Towards optimal keyword-based content dissemination in dht-based p2p networks. In: Peer-to-Peer Computing, pp. 102–111 (2011)
19. Rose, I., Murty, R., Pietzuch, P.R., Ledlie, J., Roussopoulos, M., Welsh, M.: Cobra: content-based filtering and aggregation of blogs and rss feeds. In: NSDI (2007)
20. Bianchi, P.F.S., Datta, A.K., Gradinariu, M.: Stabilizing dynamic r-tree-based spatial filters. In: ICDCS, pp. 447–457 (2007)
21. Tang, C., Dwarkadas, S.: Hybrid global-local indexing for efficient peer-to-peer information retrieval. In: NSDI, pp. 211–224 (2004)
22. Tang, C., Xu, Z., Mahalingam, M.: Psearch: information retrieval in structured overlays. In: HotNets-I (2002)
23. Tao, Y., Xiao, X., Pei, J.: Subsky: efficient computation of skylines in subspaces. In: ICDE00000, p. 65 (2006)
24. Tryfonopoulos, C., Idreos, S., Koubarakis, M.: Publish/subscribe functionality in IR environments using structured overlay networks. In: SIGIR, pp. 322–329 (2005)
25. Tryfonopoulos, C., Koubarakis, M., Drougas, Y.: Information filtering and query indexing for an information retrieval model. ACM Trans. Inf. Syst. **27**(2), 10:1–10:47 (2009)
26. Yan, T.W., Garcia-Molina, H.: The SIFT information dissemination system. ACM Trans. Database Syst. **24**(4), 529–565 (1999)