Diversity-Aware Top-k Publish/Subscribe for Text Stream

Lisi Chen
School of Computer Engineering
Nanyang Technological University
lchen012@e.ntu.edu.sg

Gao Cong
School of Computer Engineering
Nanyang Technological University
gaocong@ntu.edu.sg

ABSTRACT

Massive amount of text data are being generated by a huge number of web users at an unprecedented scale. These data cover a wide range of topics. Users are interested in receiving a few up-to-date representative documents (e.g., tweets) that can provide them with a wide coverage of different aspects of their query topics. To address the problem, we consider the Diversity-Aware Top-k Subscription (DAS) query. Given a DAS query, we continuously maintain an up-to-date result set that contains k most recently returned documents over a text stream for the query. The DAS query takes into account text relevance, document recency, and result diversity. We propose a novel solution to efficiently processing a large number of DAS queries over a stream of documents. We demonstrate the efficiency of our approach on real-world dataset and the experimental results show that our solution is able to achieve a reduction of the processing time by 60–75% compared with two baselines. We also study the effectiveness of the DAS query.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*

Keywords

text stream; diversification; publish/subscribe

1. INTRODUCTION

Massive amount of text data are being generated by a huge number of web users at an unprecedented scale. For example, Twitter, which allows user to compose tweets containing up to 140 characters, has more than 284 million monthly active users who posted 500 million tweets per day[1].

Such text data can be modeled as continuously arriving streams, and building publish/subscribe systems that can support a large number of subscribers for a text stream has many real-world applications. For example, users on Twitter want to be updated with tweets on some specific topics (e.g., food poisoning vomiting). As another example, because social updates (e.g., tweets) often offer the quickest first-hand reports of news events, comments and reviews indicating the public view, business promotion information, etc., a news website may want to annotate each news with its up-to-date relevant tweets [32]. In these applications, users would prefer to be updated with a few representative tweets rather than being overwhelmed by a large number of tweets.

Based on these applications, we consider the top-k subscription query, for which we rank-order documents and return a representative set of documents. The rationale behind is analogous to the reason that search engines rank-order documents matching a query rather than employ the boolean retrieval model (e.g., the resulting number of matching documents of a boolean filter can far exceed the number a human user could possibly sift through [26]). In fact, the top-k publish/subscribe query that ranks documents based on their relevance to query keywords has been studied in recent years (e.g., [18,30,32]). The state-of-the-art top-k publish/subscribe system for tweets [32] returns a subscriber top-k tweets that are ranked based on both keyword relevance and tweet recency.

However, a major problem of these top-k publish/subscibe systems is that they do not consider the diversification of results. The problem of result diversification has drawn considerable attention as a way to increase user satisfaction in recommendation systems and web search [11]. Similarly, diversifying the top-k subscription result will provide users a better coverage of different aspects of the query. Moreover, queries are sometimes ambiguously or vaguely specified. For example, queries issued on Twitter only contain 1.64 keywords on average [34], which indicates the need for result diversification. Note that relevance, recency, and diversity are considered in existing work on tweet search [7, 32].

Taking into account diversity in top-k publish/subscribe system incurs new challenges. First, it is an open problem to define the diversity-aware subscription query. On the one hand, we should consider all the three aspects in the definition. On the other hand, the definition should allow us to develop efficient algorithm to handle a large number of subscription queries. To address the challenge, we propose a new type of top-k subscription query, which is referred to as Diversity-Aware Top- k Subscription (DAS) query. The DAS query takes into account the following three aspects in evaluating its results: (1) Text relevance; (2) Document recency; (3) Result diversity. DAS queries are treated as subscriptions and documents from a text stream are published items in the publish/subscribe system. Each DAS query maintains a result set that contains k most recently returned documents for the query over a stream of documents. A DAS query is triggered by a new published document

[1]https://about.twitter.com/company (accessed date: 6 Nov, 2014)

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

http://dx.doi.org/10.1145/2723372.2749451.

only if updating its result set with the new document increases the diversity and relevance score of the current result set. The diversity and relevance score is computed by considering all the three aspects.

The second challenge is that the proposed publish/subscribe system should efficiently support millions of subscription queries and determine whether each query can be triggered by a new document. A straightforward approach would work as follows: given a new published document d_n, for each DAS query q we compute the diversity and relevance score of the result set with the oldest document replaced by d_n. If the score is larger than the diversity and relevance score of the current result set, document d_n becomes a result, replacing the oldest document in $q.R$; otherwise d_n is not a result. Because every document in each query result set is involved in determining whether a new document can be a new result, it is rather costly to process a large number of DAS queries. This calls for an effective filtering technique to make the publish/subscribe system efficient. To address the challenge, we propose a novel approach including the following key techniques.

(1) We propose the concepts of *individual filtering condition* and *group filtering condition* to help determine whether a DAS query and a group of DAS queries can be filtered out by each new document, respectively.

(2) We develop the *group filtering technique* to efficiently compute group filtering condition. Specifically, we propose the concept of *minimal covering set* (MCS) and define the MCS *maximization problem* to find a set of MCSs for utilizing the documents that are shared by the results of different queries in a group. Then we devise an efficient algorithm to solve the problem and based on the MCS we develop a technique to efficiently estimate the group filtering condition.

(3) We develop the *individual filtering technique* to further improve the efficiency of generating individual filtering condition.

In summary, the paper's contributions are threefold.

First, we define the DAS query, which considers diversity, relevance, and recency. We build the first diversity-aware top-k publish/subscribe system for a text stream, which aims to maintain the up-to-date results for a large number of DAS queries over a text stream.

Second, we propose a novel approach comprising the aforementioned key techniques that is capable of processing a large number of DAS queries efficiently.

Third, we conduct an extensive experimental study for evaluating the paper's proposals on real-world datasets of a large scale collected from Twitter. Our experiments demonstrate that our proposed filtering techniques are able to improve the runtime performance by 60% to 75% compared with two baselines developed by us. Our user study result demonstrates that the DAS query produces results with comparable quality in comparison to the queries defined by two state-of-the-art diversity-aware systems [12, 27], which are developed for processing a single query, rather than a large number of subscription queries. Moreover, our experimental results show that the existing diversity-aware systems [12, 27] fail to work for a set of 2M subscription queries, while our approach can efficiently support them. Our system runs faster than the two state-of-the-art diversity-aware systems by more than an order of magnitude for 100K subscription queries.

The rest of this paper is organized as follows. Section 2 defines the DAS query. Section 3 presents an overview of the proposed solution. Section 4 introduces our methods for representing and indexing DAS queries, and the concepts of individual filtering condition and group filtering condition. Based on the filtering conditions we detail the group filtering technique and individual filtering technique in Section 6 and Section 7, respectively. Then we present the experimental studies in Section 8. Section 9 reviews the related work, and Section 10 concludes the paper. Proofs, baselines, and additional experimental results are put in Appendix.

2. PROBLEM STATEMENT

We introduce the text stream and define the Diversity-Aware Top-k Subscription (DAS) query.

Definition 1: Text Stream. A text stream comprises a sequence of text documents, each denoted by a triple $d = \langle id, v_d, t_c \rangle$, where id is the document id, which is assigned based on the creation time of d, v_d is a sequence of words from the vocabulary $V = \{w_1, w_2, ..., w_{|V|}\}$, and t_c indicates the creation time of d. □

The text documents in Definition 1 can be tweets in Twitter, posts and comments on Facebook, etc.

Intuitively, given a stream of text, Diversity-Aware Top-k Subscription (DAS) query is to continuously maintain the result set over time, which contains top-k ranked documents based on their diversity and relevance scores. The following is considered in the diversity and relevance score: (1) *Relevance*, which is the text relevance between the document and the query keywords. The DAS query favors the documents that are relevant to the query keywords; (2) *Diversity*, which is measured by the sum of pair-wise similarities between any two documents in the query result set (the lower value indicates the higher degree of diversity); (3) *Recency*, which is computed based on the arrival time of a document. The DAS query favors the newly arrived documents. Note that recency is important for data streams. According to a comparison between microblog search and web search [34], the intent for microblog search is always inclined to find more fresh information. Some other literatures also claim that tweets are often tied to some specific event and their relevance to a query declines as time passes (e.g., [32,37]).

Definition 2: Diversity-Aware Top-k Subscription (DAS) Query. A DAS query is represented by a tuple $q = \langle id, \psi \rangle$, where id indicates the query id, and ψ is a set of query keywords. We use $q.R$ to denote the result set of q, where $|q.R| = k$.

A DAS query aims to continuously feed the user with new document d_n that can be ranked within the top-k based on the following criteria:

(1) If d_n does not contain any keyword in $q.\psi$, d_n cannot become a result of q.

(2) Let d_e be the document with the earliest time of arrival in $q.R$. If by replacing d_e with d_n, the diversity and relevance score of $q.R$ becomes larger, then d_n will replace d_e as the new result of q; otherwise d_n cannot be a result of q. □

We define the diversity and relevance score based on the *max-sum diversification*, which is a natural bi-criteria objective function that considers both the relevance and dissimilarity of a selected set [16]. Such function is widely applied and studied for result diversification (e.g., [5, 16, 27]). Specifically, the diversity and relevance score of query q, denoted by $DR(q.R)$, is computed as follows:

$$DR(q.R) = \alpha \times \sum_{d_i \in q.R} R(q, d_i) + (1 - \alpha) \times D(q.R), \qquad (1)$$

where $R(q, d)$ computes the relevance score between query q and document d_i, $D(q.R)$ computes the diversity score of documents in $q.R$, and $\alpha \in [0, 1]$ is a system parameter that specifies the trade-off between relevance and diversity.

Relevance: Following the previous work (e.g., [32]), we compute the score of relevance between query q and document d by combining the score of text relevance and a temporal decaying factor,

which is as follows:

$$R(q,d) = TRel(q,d) \times T(d), \tag{2}$$

We use language models to compute the score of text relevance $TRel(q,d)$, which is described as follows:

$$TRel(q,d) = \prod_{w \in q.\psi} PS(d.v_d, w), \tag{3}$$

where $PS(d.v_d, w)$ is the text relevance of term w to $d.v_d$, which is computed as follows:

$$PS(d.v_d, w) = (1-\lambda)\frac{Num(d.v_d, w)}{|d.v_d|} + \lambda \frac{Num(Coll, w)}{|Coll|},$$

where $Num(d.v_d, w)$ is the number of occurrences of w in $d.v_d$, $Num(Coll, w)$ represents the number of occurrences of w in the document collection $Coll$, and λ is a smoothing parameter of the Jelinek-Mercer smoothing method.

Note that other measurements of text relevance (e.g., cosine similarity) can also be applied in our proposed method.

Recency: The recency of document d, which is denoted by $T(d)$, is calculated by the exponential decay function, namely:

$$T(d) = B^{-(t_{cur} - d.t_c)}, \tag{4}$$

where B is base number that determines the rate of the recency decay, t_{cur} indicates the current time, and $d.t_c$ refers to the creation time of d. The function is monotonically decreasing with $t_{cur} - d.t_c$. It is introduced in [23] and is applied (e.g., [2, 24, 32]) as the measurement of recency for stream data. Based on the experimental studies [14], the exponential decay function has been shown to be effective in blending the recency and text relevancy of documents.

Diversity: Given a query result set $q.R$, we compute the diversity score of $q.R$ following the max-sum objective function, which is widely used for result diversification (e.g., [5, 16, 27]), by summing up the dissimilarity score of any pairs of documents in $q.R$, which is defined as follows:

$$D(q.R) = \frac{2}{k-1} \sum_{d_i, d_j \in q.R} \mathsf{d}(d_i, d_j), \tag{5}$$

where

$$\mathsf{d}(d_i, d_j) = 1 - Sim(d_i, d_j) \tag{6}$$

and $Sim(d_i, d_j)$ denotes the cosine similarity between $d_i.v_d$ and $d_j.v_d$, Note that $\frac{2}{k-1}$ is used for balancing out the fact that there are $\frac{k-1}{2}$ pairs for each document in the similarity sum.

We aim to develop a scalable solution to maintain the up-to-date results for a large number of DAS queries over a stream of text. Millions of DAS queries can easily fit into the available memory of modern servers. Hence, our solution is developed under this setting. In the case that the DAS queries cannot fit into memory, we can employ our proposed solution on multiple servers, each handling a subset of DAS queries independently.

3. FRAMEWORK OVERVIEW

We aim at addressing the problem of answering a large number of DAS queries over text streams. A straightforward approach would work as follows: given a new published document d_n, for each DAS query q we compute the diversity and relevance score of $q.R$ with d_e replaced by d_n (d_e represents the document with the earliest time of arrival in $q.R$). If the diversity and relevance score is larger than the diversity and relevance score of $q.R$, document d_n becomes a result and is used to replace d_e in $q.R$; otherwise d_n is not a result. Note that for each new document d_n we need to compute the diversity and relevance score with respect to each query result set $q.R$, and for each query result set we need to separately evaluate the similarity between d_n and each document $q.R \backslash \{q.d_e\}$. Therefore, the approach is computationally expensive especially when the number of queries is large or the documents arrive at a high rate. Hence, we need a more efficient mechanism to handle DAS queries over the data stream, where both new queries and new documents arrive in a streaming manner.

An underlying idea of many publish/subscribe systems (e.g., [10, 32]) is to group subscription queries such that they can be evaluated simultaneously for a new published document. Motivated by these systems, we also expect to design an approach to grouping DAS queries and DAS query results such that queries in one group can be evaluated simultaneously, thus reducing the computation of query processing. Specifically, we propose the concepts of *query filtering condition* and *group filtering condition* respectively for filtering out the documents that cannot be the result for any query. Based on the concepts we group and index the DAS queries by the block based inverted file and we develop a two-stage filtering mechanism, which is briefed as follows.

(1) *Group filtering technique*, which is used for checking whether a new document can be a result of some query in a query block based on the group filtering condition. We propose the concept of *minimal covering set* to help generate the group filtering condition for each query block. Then we develop an efficient algorithm to derive the minimal covering sets based on the results of queries in each block. With the generated minimal covering sets, we develop an approach to handling queries in a block simultaneously for a new document. This technique will be presented in Section 5.

(2) *Individual filtering technique*, which is utilized when a query cannot be filtered during the group filtering stage. We propose the technique to optimize the filtering efficiency while taking into account the available memory space. This technique will be presented in Section 6.

Figure 1 illustrates our proposed architecture for processing DAS queries. Blue arrows denote the process of a queries and green arrows denote the process of documents. A user may both issue a query and generate a document.

When the system receives a DAS query, the query is firstly initialized by traversing the *document lists*, then it is inserted into the *query inverted file* and the *query result index*. The query inverted file consists of *query postings lists* and *group-based query result summaries*. Each query postings list corresponds to a term, and the list stores the ids of the queries that contains the term. The group-based query result summaries store the minimal covering sets for each query block. The query result index consists of *query result tables* and *aggregated term weight summaries* that maintain the result of each query.

When a new document d_n arrives, it is stored in the document lists that stores the text and temporal information of each arrived document. Then we regard d_n as "query" and based on the group filtering technique we traverse the query inverted lists and the group-based query result summaries to filter d_n over the blocks of query inverted file. If a block cannot filter d_n, we visit the query result tables and the aggregated term weight summaries for each query q in the block to determine whether q can include d_n as their results. Finally, d_n are pushed to the users who issue the queries that can include d_n as their result.

Table 1 presents the notations frequently used in the paper.

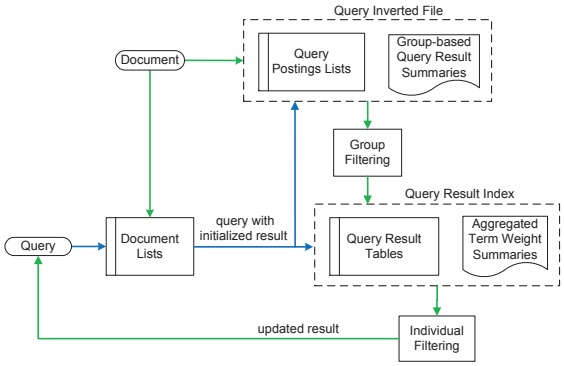


Figure 1: Architecture for Processing DAS Queries

Notation	Definition
$q.R$	result of query q
d_n	a new document
$q.d_e$	document with the earliest creation time in $q.R$
$q.R'$	$q.R \cup \{d_n\} \setminus \{d_e\}$
G	a group of queries
q_e	query with earliest d_e among all queries in a group
b	a block of query postings list
$\mathsf{dr}_q(d_n)$	div and rel score of q contributed by d_n
$\mathsf{dr}_q(q.d_e)$	div and rel score of q contributed by $q.d_e$
$\mathsf{TRel}_{max}(G, d_n)$	$max\{TRel(q, d_n)\|q_i \in G\}$
$\mathsf{Sim}_{min}(G, d_n)$	$min\{\sum_{d_j \in q_i.R' \setminus \{d_n\}} Sim(d_j, d_n)\|q_i \in G\}$
$U_w(b)$	universe of block b in postings list of w
$Q_s(b, d_i)$	a set of queries in b that contain d_i in their results
$\mathsf{AW}(w_i, S_d)$	aggregated term weight of w_i w.r.t. S_d

Table 1: Summary of Notations

4. REPRESENTING AND INDEXING DAS QUERIES

We introduce our proposed method of representing and indexing DAS queries, which lays the foundation of our group filtering technique and individual filtering technique. We first propose the concept of *individual filtering condition* to represent each DAS query (Section 4.1). Based on the individual filtering condition, we develop an approach to deriving a *group filtering condition* for a group of queries, and show how to use the group filtering condition to determine whether a new document can be a result of some query in a group (Section 4.2). Finally we develop an approach to indexing DAS queries (Section 4.3).

4.1 Individual Filtering Condition

We propose the concept of individual query filtering condition to represent the DAS query, which is used for determining whether a new document can be a result of a query.

We use d_n to denote a new document, q to denote a DAS query, $q.d_e$ to denote the document with the earliest arrival time in $q.R$, and $q.R'$ to denote an updated $q.R$ where d_e is replaced by d_n (i.e., $q.R' = q.R \cup \{d_n\} \setminus \{q.d_e\}$). Based on Definition 2, to check whether d_n can be a result of q we need compare $DR(q.R)$ and $DR(q.R')$. However, it is time-consuming to re-compute the two values each time when a new document arrives (the complexity of the re-computation is $O(k^2)$ for each query). We need a more efficient way to compare the two values.

Specifically, we find that the comparison between $DR(q.R)$ and $DR(q.R')$ can be deduced to an equivalent comparison that is much more efficient. The details are presented in Lemma 1.

Let $\mathsf{dr}_q(q.d_e)$ be the diversity and relevance score of q contributed by $q.d_e$, $\mathsf{dr}_q(d_n)$ be the diversity and relevance score of q contributed by a new document d_n. We compute $\mathsf{dr}_q(q.d_e)$ and $\mathsf{dr}_q(d_n)$ as follows.

$$\mathsf{dr}_q(q.d_e) = \alpha \times R(q, q.d_e) + \frac{2-2\alpha}{k-1} \sum_{d_i \in q.R \setminus \{q.d_e\}} \mathsf{d}(q.d_e, d_i). \quad (7)$$

$$\mathsf{dr}_q(d_n) = \alpha \times R(q, d_n) + \frac{2-2\alpha}{k-1} \sum_{d_i \in q.R' \setminus \{d_n\}} \mathsf{d}(d_n, d_i). \quad (8)$$

Lemma 1: Given a new document d_n, we have:

$$DR(q.R') - DR(q.R) = \mathsf{dr}_q(d_n) - \mathsf{dr}_q(q.d_e). \quad (9)$$

Proof Sketch: The proof can be found in the Appendix. □

Based on Lemma 1, we just need compare $\mathsf{dr}_q(d_n)$ and $\mathsf{dr}_q(q.d_e)$ for determining whether d_n can be a result of q, and the complexity is $O(k)$. This is much more efficient than computing $DR(q.R)$ and $DR(q.R')$.

From Equation 2, we observe that $\mathsf{dr}_q(q.d_e)$ is independent of the contents of new documents, while $\mathsf{dr}_q(d_n)$ depends on the new documents. Hence we use $\mathsf{dr}_q(q.d_e)$ as the filtering threshold to represent the DAS query q. Specifically, for a new document d_n, if $\mathsf{dr}_q(d_n) > \mathsf{dr}_q(q.d_e)$, then d_n is a result of q; otherwise d_n cannot be a result of q.

The filtering condition of a DAS query is defined as follows:

Definition 3: Filtering Condition of DAS Query: Given a DAS query q and a new document d_n, the filtering condition of q w.r.t. d_n is:

$$\mathsf{dr}_q(d_n) \leq \mathsf{dr}_q(q.d_e). \quad (10)$$

Here $\mathsf{dr}_q(q.d_e)$ is independent of d_n and it is called *filtering threshold*. If Inequation 10 is satisfied, then d_n is filtered out. □

4.2 Group Filtering Condition

Section 4.1 introduces how to derive the filtering condition of a DAS query. We proceed to present how to generate filtering condition for a group of queries, which lays the foundation of our approach to filtering new documents simultaneously for a group of queries.

Let $G = \{q_0, q_1, ..., q_{n-1}\}$ be a group of n DAS queries. Based on Equation 10 we can obtain a filtering threshold of G by computing the minimum filtering threshold for all queries in G, i.e., $min\{\mathsf{dr}_{q_i}(q_i.d_e)|q_i \in G\}$. If a new document d_n satisfies the following condition, then d_n is filtered out by G.

$$max\{\mathsf{dr}_{q_i}(d_n)|q_i \in G\} \leq min\{\mathsf{dr}_{q_i}(q_i.d_e)|q_i \in G\} \quad (11)$$

Inequation 11 is the exact group filtering condition for G, where $min\{\mathsf{dr}_{q_i}(q_i.d_e)|q_i \in G\}$ is the corresponding filtering threshold. However, both of the two values are computationally expensive for each group G, which is $O(k|G|)$. We develop more efficient ways to compute a lower bound of filtering threshold of G and an approximate value of $max\{\mathsf{dr}_{q_i}(d_n)|q_i \in G\}$ respectively to expedite the checking of whether a new document can be a result of some queries in G.

4.2.1 Lower bound of group filtering threshold

Since $\mathsf{dr}_{q_i}(q_i.d_e)$ depends on time, the filtering threshold of G is also time-dependent. Therefore, the exact value of filtering threshold for G need be updated by re-computing $\mathsf{dr}_{q_i}(q_i.d_e)$ for each q_i in G when a new document arrives. Such frequent re-computations are computationally expensive.

To avoid the expensive re-computation, we propose a method to estimate a lower bound of filtering threshold for G, which is

denoted by $\widetilde{\mathsf{FT}}_G$:

$$\widetilde{\mathsf{FT}}_G = DTRel_{min}(G) - \alpha \times TRel(q_m, q_m.d_e) \times (1 - T(q_e.d_e)), \tag{12}$$

where

$$DTRel_{min}(G) = min\{\alpha \cdot TRel(q_i, q_i.d_e) + \frac{2-2\alpha}{k-1} \sum_{d_j \in q_i.R \setminus \{q_i.d_e\}} \mathsf{d}(d_j, q_i.d_e) | q_i \in G\}, \tag{13}$$

q_m is the query in G such that

$$TRel(q_m, q_m.d_e) = max\{TRel(q_i, q_i.d_e) | q_i \in G\}, \tag{14}$$

and q_e denotes the query with earliest d_e among all queries in G. Note that $TRel(q_m, q_m.d_e)$, $DTRel_{min}(G)$, and $q_e.d_e$ are time-independent, so we can store them and then when a new document arrives the complexity of deriving $\widetilde{\mathsf{FT}}_G$ is only $O(1)$, rather than $O(k|G|)$ as discussed earlier.

Lemma 2 establishes the correctness of $\widetilde{\mathsf{FT}}_G$.

Lemma 2: Given a set of DAS queries G, we always have:

$$\widetilde{\mathsf{FT}}_G \leq min\{\mathsf{dr}_{q_i}(q_i.d_e) | q_i \in G\}. \tag{15}$$

Proof Sketch: The proof can be found in the Appendix. □

4.2.2 *Computing* $max\{\mathsf{dr}_{q_i}(d_n) | q_i \in G\}$

The exact value of $max\{\mathsf{dr}_{q_i}(d_n) | q_i \in G\}$ can be computed only by calculating each $\mathsf{dr}_{q_i}(d_n)$, which is also time-consuming. So the next problem is how to derive an approximate value (upper bound) of $max\{\mathsf{dr}_{q_i}(d_n) | q_i \in G\}$ without checking each query in G and its results.

We first introduce two notations: $\mathsf{TRel}_{max}(G, d_n)$ represents the maximum text relevance between d_n and the queries in G, which is defined by Equation 16, and $\mathsf{Sim}_{min}(G, d_n)$ refers to the minimum sum of similarities between d_n and the results of queries in G, which is defined by Equation 17.

$$\mathsf{TRel}_{max}(G, d_n) = max\{TRel(q, d_n) | q_i \in G\} \tag{16}$$

$$\mathsf{Sim}_{min}(G, d_n) = min\{\sum_{d_j \in q_i.R' \setminus \{d_n\}} Sim(d_j, d_n) | q_i \in G\}. \tag{17}$$

The way to compute an approximate $max\{\mathsf{dr}_{q_i}(d_n) | q_i \in G\}$ is established in Lemma 3.

Lemma 3: Given a set of DAS queries G, we always have:

$$max\{\mathsf{dr}_{q_i}(d_n) | q_i \in G\} \leq \alpha \times \mathsf{TRel}_{max}(G, d_n) + \frac{2-2\alpha}{k-1}(k - 1 - \mathsf{Sim}_{min}(G, d_n)).$$

Proof Sketch: The proof can be found in the Appendix. □

The remaining problem is how to compute $\mathsf{Sim}_{min}(G, d_n)$ and $\mathsf{TRel}_{max}(G, d_n)$, which is very important for the efficiency and is respectively discussed in Section 5 and Section 7.

4.3 Indexing DAS Queries

Before introducing the approaches to computing $\mathsf{TRel}_{max}(G, d_n)$ and $\mathsf{Sim}_{min}(G, d_n)$, we first present how to index the DAS queries.

We organize the DAS queries into an inverted file, which can help prune the queries that do not share any keywords with a new document. The inverted file consists of postings lists, each of which is associated with a term w and comprises a sequence of postings. Each posting contains the identifier of a query q that contains term w. With the inverted file, we are able to discard the queries that do not share any common term with the new document. Recall that according to Definition 2 if a new document does not contain any query keyword of a DAS query, the document will not be a result.

Specifically, we use the block based inverted file to index the DAS queries. As illustrated in Figure 2, in the block based inverted file, each postings list consists of a number of postings block, each of which contains at most p_{max} postings, where p_{max} is a system parameter. Each posting stores a query id, and the postings in each postings list are sorted in ascending order of their query ids.

To enable group filtering under the block based inverted file, we augment each block b_i with the following components:

(1) $minID$ and $maxID$, which respectively indicates the minimum and maximum ids of postings (queries) in b_i;

(2) $DTRel_{min}(b_i)$ (Equation 13);

(3) $TRel(q_m, q_m.d_e)$ (Equation 14);

(4) $q_e.d_e$, which denotes the earliest d_e of queries in b_i;

(5) Result summary of b_i, which summarizes the results of queries in b_i with minimal covering sets. This is a key component for our proposed group filtering technique, and will be introduced in Section 5.

To enable individual filtering for each query, we also index the results of each query with a query result index, which will be detailed in Section 6.

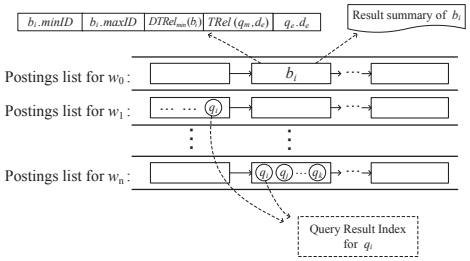


Figure 2: Query Inverted File

5. GROUP FILTERING TECHNIQUE

We proceed to present the group filtering technique based on the query inverted file.

At the beginning, we briefly introduce our approach to processing DAS queries when a new document arrives. The high-level idea is as follows. For a new document d_n we traverse the query postings lists of all the terms contained in $d_n.v_d$ simultaneously based on the Document-at-a-Time (DAAT) technique [26]; for each postings list we maintain a cursor that specifies the query id we currently visit. Before visiting the postings (queries) in a new block b, we apply group filtering condition to check whether b can be filtered without evaluating each individual query in b.

Recall that the group filtering condition for block b consists of three components, namely $\mathsf{TRel}_{max}(b, d_n)$, $\mathsf{Sim}_{min}(b, d_n)$, and $\widetilde{\mathsf{FT}}_b$. We first present how to estimate $\mathsf{TRel}_{max}(b, d_n)$, and then we focus on how to efficiently compute $\mathsf{Sim}_{min}(b, d_n)$, which is the most challenging problem in the group filtering stage.

Estimating $\mathsf{TRel}_{max}(b, d_n)$. Based on the postings lists and the DAAT technique, we compute an upper bound of $\mathsf{TRel}_{max}(b, d_n)$ as follows. Let p_w be the current position of the cursor in the postings list of w. An upper bound of $\mathsf{TRel}_{max}(b, d_n)$, which is denoted by $\widetilde{\mathsf{TRel}}_{max}(b, d_n)$, is computed as follows:

$$\widetilde{\mathsf{TRel}}_{max}(b, d_n) = max\{PS(d_n.v_d, w_i) | w_i \in d_n.\psi \wedge p_{w_i} \leq b.maxID\}. \tag{18}$$

Lemma 4 establishes the correctness of $\widetilde{\mathsf{TRel}}_{max}(b, d_n)$.

Lemma 4: Given a new document d_n and a block b, we always have:

$$\widetilde{\mathsf{TRel}}_{max}(b, d_n) \geq \mathsf{TRel}_{max}(b, d_n).$$

Proof Sketch: The proof can be found in the Appendix. □

We proceed to present how to compute $\mathsf{Sim}_{min}(b, d_n)$. From Equation 17, we observe that we have to evaluate each $q_i \in b$ to compute the value of $\mathsf{Sim}_{min}(b, d_n)$. However, it is time-consuming to evaluate each query in a block for computing $\mathsf{Sim}_{min}(b, d_n)$. Therefore, the challenge here is how to compute an approximate value of $\mathsf{Sim}_{min}(b, d_n)$ without the need of evaluating each query in the block.

To solve the problem, we propose the concept of minimal covering set (MCS) and we define the MCS maximization problem that aims at generating maximum number of minimal covering sets for each block b_i. Then we develop an efficient algorithm for MCS maximization problem. Based on the minimal covering sets for b_i, we develop an approach to computing an approximate value of $\mathsf{Sim}_{min}(b_i, d_n)$ without evaluating every query in b_i.

5.1 MCS Maximization Problem

Before introducing the MCS maximization problem, we present the concept of minimal covering set.

Recall that we want to compute an approximate $\mathsf{Sim}_{min}(b, d_n)$ without checking the result of each query in b. However, it is difficult to achieve this because each query in b has different results. Nevertheless, we observe that some queries in b may share some common documents in their results. Therefore, we propose the concept of minimal covering set for representing a set of documents that are shared by the results of queries in b.

We first present the concept of *universe*.

Definition 4: Universe of a block. Let $b = \{q_0, q_1, ..., q_{n-1}\}$ be a block in the query postings list of term w. The universe of b, denoted by $U_w(b)$, satisfies the following conditions:

(1) $U_w(b) \subseteq \bigcup_{q_i \in b}\{q_i.R \setminus \{q_i.d_e\}\}$;

(2) For each d_i in $U_w(b)$, $w \in d_i.v_d$. □

Note that the document with the earliest arrival time in each query result set will not be considered for matching new documents. The reason that it will be removed when the query result is updated by a new document so that it will not induce any change w.r.t. the diversity score of the query. The minimal covering set is defined as follows.

Definition 5: Minimal Covering Set (MCS). Let $Q_s(b, d_i)$ be $\{q_j | q_j \in b \wedge d_i \in q_j.R \setminus \{q_j.d_e\}\}$, which is a set of queries in b that contain d_i in their results. A set of documents S is called a minimal covering set (MCS) of queries in block b if the following two conditions are satisfied:

(1) $\bigcup_{d_i \in S} Q_s(b, d_i) = b$;

(2) $\forall d_i \in S$, $S \setminus \{d_i\}$ does not satisfy (1). □

In Definition 5, condition (1) suggests that given an MCS S, the result of each query in b contains at least one document in S as its result, and condition (2) requests MCS be the minimal set that satisfies condition (1).

We can generate a number of MCSs based on $U_w(b)$ and the queries in b. Let $\mathfrak{S}_w(b)$ be the set of disjoint MCSs generated from $U_w(b)$. For a new document d_n, we can estimate an approximate value of $\mathsf{Sim}_{min}(b, d_n)$ (denoted by $\widetilde{\mathsf{Sim}}_{min}(b, d_n)$) by Equation 19.

$$\widetilde{\mathsf{Sim}}_{min}(b, d_n) = \sum_{S_i \in \mathfrak{S}_w(b)} min\{Sim(d_n, d_j) | d_j \in S_i\} + minSim(U_w(b), d_n) \times (k - |\mathfrak{S}_w(b)|), \quad (19)$$

where $minSim(U_w(b), d_n)$ is the minimum possible similarity between d_n and any documents in $U_w(b)$.

We next present how to compute $minSim(U_w(b), d_n)$. Let w be the term of the postings list that b belongs to and $d_i.v_d.w$ be the term frequency of w in $d_i.v_d$. $minSim(U_w(b), d_n)$ is computed by Equation 20.

$$minSim(U_w(b), d_n) = \frac{min\{d_i.v_d.w \mid d_i \in U_w(b)\} \times d_n.v_d.w}{max\{\| d_i.v_d \| | d_i \in U_w(b)\} \times \| d_n.v_d \|}. \quad (20)$$

Note that both $min\{d_i.v_d.w \mid d_i \in U_w(b)\}$ and $max\{\| d_i.v_d \| | d_i \in U_w(b)\}$ are independent of the content of the new document d_n. As a result, if we index the above two values for each block, then the complexity of computing $minSim(U_w(b), d_n)$ is $O(1)$.

We proceed to demonstrate the effectiveness of MCS. Since $min\{Sim(d_n, d_j) | d_j \in S_i\}$ relies on the content of d_n, we are unable to compute it before d_n arrives. Here we assume that the expected value of $min\{Sim(d_n, d_j) | d_j \in S_i\}$ for each S_i is e_s. Based on this assumption, we propose Lemma 5 to demonstrate that $\widetilde{\mathsf{Sim}}_{min}(b, d_n)$ is affected by the number of MCSs in $\mathfrak{S}_w(b)$.

Lemma 5: Let $\mathfrak{S}_w(b)$ and $\mathfrak{S}'_w(b)$ be two sets of pairwise disjoint MCSs based on the universe $U_w(b)$ s.t. $|\mathfrak{S}_w(b)| \geq |\mathfrak{S}'_w(b)|$, and $min\{Sim(d_n, d_j) | d_j \in S_i\} = e_s$ for each S_i. Assume that $\widetilde{\mathsf{Sim}}_{min}(b, d_n)$ is computed by Equation 19 based on $\mathfrak{S}_w(b)$, and $\widetilde{\mathsf{Sim}}'_{min}(b, d_n)$ is computed by Equation 19 based on $\mathfrak{S}'_w(b)$, then we have:

$$\widetilde{\mathsf{Sim}}_{min}(b, d_n) \geq \widetilde{\mathsf{Sim}}'_{min}(b, d_n). \quad (21)$$

Proof Sketch: The proof can be found in the Appendix. □

Example 1 illustrates the concept of MCS.

Example 1: Let b be a block containing 8 postings (queries) in the query postings list and $U_w(b)$ be $\{d_1, d_2, ..., d_9\}$. For each query in b the corresponding results in $U_w(b)$ are shown in Figure 3. The id of a document is assigned based on its arrival time and the results of each query are sorted in descending order of their document ids. Note that we exclude d_0 from $U_w(b)$ since d_0 is the document with the earliest arrival time in the results of each query. For better illustrating the MCS, we represent the result of each query by Table 2, where "X" denotes that the corresponding document is a result of the corresponding query.

From Table 2, we can find that $S_0 = \{d_1\}$ is an MCS because all queries in b have d_1 as their results, $S_1 = \{d_4, d_5\}$ is also an MCS because all the queries in b contains either d_4 or d_5 in their results, while $S_2 = \{d_6, d_7\}$ cannot be an MCS since q_3 is not covered by S_2, and $S_3 = \{d_3, d_4, d_6\}$ cannot be an MCS either because there exists one or more MCSs that belong to the subsets of S_3 (e.g., $\{d_3, d_4\}$). □

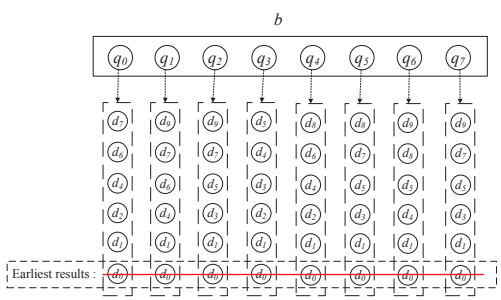


Figure 3: Results of Queries in Block b

	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7
d_1	X	X	X	X	X	X	X	X
d_2	X			X	X			
d_3			X	X		X		X
d_4	X	X	X	X	X		X	
d_5				X		X	X	X
d_6	X	X			X			
d_7	X	X	X			X		X
d_8	X				X	X	X	
d_9		X	X				X	X

Table 2: Inverted Lists for Query Results in b

Lemma 5 suggests that to obtain a tight estimation of $\mathsf{Sim}_{min}(b, d_n)$ we aim at maximizing the number of disjoint MCSs (i.e., $|\mathfrak{S}_b|$) for each block b. We define MCS maximization problem as follows.

Definition 6: MCS Maximization Problem: Let $b = \{q_1, q_2, ..., q_n\}$ be a block of n DAS queries, and $U_w(b) = \{d_1, d_2, ..., d_m\}$ be the universe of b. The MCS maximization problem is to find a set of MCSs $\mathfrak{S}_w(b)$ with the maximum cardinality such that (1)$(\forall S_i \in \mathfrak{S}_w(b))S_i \subseteq U_w(b)$; (2) $(\forall S_i, S_j \in \mathfrak{S}_w(b))S_i \cap S_j = \emptyset$. □

Theorem 1: The MCS maximization problem is NP-hard.

Proof Sketch: The proof can be found in the Appendix. □

Note that we do not compute a new set of MCSs for b (i.e.,$\mathfrak{S}_w(b)$) whenever a new document becomes a result of some queries in b. Instead, $\mathfrak{S}_w(b)$ may be used by multiple new documents. We discuss the update of $\mathfrak{S}_w(b)$ in Section 7.1.

5.2 Algorithm for MCS Maximization

Since MCS maximization problem is NP-hard, computing the exact result of $\mathfrak{S}_w(b)$ is computationally prohibitive. We develop a greedy algorithm with a $(s_{max}/2 + \epsilon)$ approximation ratio where s_{max} refers to the maximum cardinality of the MCS generated from $U_w(b)$ and ϵ refers to an arbitrarily small value.

The high-level idea of the algorithm is to greedily select the document from $U_w(b)$ that is included as a result by the largest number of queries as their results to generate a MCS.

Algorithm 1: GreedyMcsGen **(Block** b**, Universe** $U_w(b)$**,** $\{Q_s(b, d_i)|d_i \in U_w(b)\}$**)**

```
1  S_w(b) ← ∅;
2  U_r ← U_w(b);
3  while U_r is not empty do
4  |  S_r ← ∅;
5  |  Q_r ← all queries in b;
6  |  while Q_r is not empty do
7  |  |  d_m ← NULL;
8  |  |  c_max ← 0;
9  |  |  for each d_i ∈ (U_r \ S_r) do
10 |  |  |  if |Q_s(d_i) ∩ Q_r| > c_max then
11 |  |  |  |  c_max ← |Q_s(d_i) ∩ Q_r|;
12 |  |  |  |  d_m ← d_i;
13 |  |  S_r.add(d_m);
14 |  |  U_r.remove(d_m);
15 |  |  if U_r is empty then
16 |  |  |  break;
17 |  |  Q_r ← Q_r \ Q_s(d_m);
18 |  S_w(b).add(S_r);
19 return S_w(b);
```

Algorithm 1 shows the pseudo code. It uses the following variables: U_r is used for storing the documents in $U_w(b)$ that are not returned as an element in MCS yet, S_r is used for maintaining the current MCS under generation, and Q_r is used for storing the current set of queries that do not contain any document in S_r as their results. We initialize U_r to be $U_w(b)$ (line 2), S_r to be empty (line 4), and Q_r to be the queries in b (line 5). Next, for generating a MCS we first choose an document d_m from U_r that is included as a result by the largest number of queries (lines 9-12). Then we add the chosen document d_m into S_r (line 13), remove d_m from U_r (line 14), and remove the queries of which results contain d_m from Q_r (line 17). The process is repeated until Q_r is empty, indicating that an MCS is generated (i.e. S_r). When an MCS is generated, we add S_r into $\mathfrak{S}_w(b)$ (line 18). The algorithm terminates when U_r becomes empty.

We proceed to analyze the approximation ratio of GreedyMcsGen. We denote the optimal number of disjoint MCSs by R_{opt}, and the number of disjoint MCSs found by GreedyMcsGen by R_G. The cardinality of the MCS with the maximum number of documents is denoted by s_{max}. We have the following theorem:

Theorem 2: $(s_{max}/2 + \epsilon)R_G \geq R_{opt}$, where ϵ is an arbitrary small value.

Proof Sketch: The proof can be found in the Appendix. □

6. INDIVIDUAL FILTERING TECHNIQUE

For a new document d_n and a block b, if the queries in b cannot filter out d_n in the group filtering stage, we need separately evaluate each query in b for checking whether d_n is a result of the query. In this section, we present the technique for evaluating each query based on the query filtering condition introduced in Section 4.1.

According to Equation 10, we need calculate $\mathsf{dr}_q(d_n)$ and compare it with $\mathsf{dr}_q(q.d_e)$ to determine whether new document d_n can replace d_e in $q.R$. If $q.d_e$ is replaced by d_n, we update $\mathsf{dr}_q(q.d_e)$ because the document with the earliest arrival time in $q.R$ is changed.

We present how to compute $\mathsf{dr}_q(d_n)$ in Section 6.1 and then we present the structure of query result index in Section 6.2.

6.1 Computation of $\mathsf{dr}_q(d_n)$

For computing $\mathsf{dr}_q(d_n)$, according to Equation 7 we need compute (1) $TRel(q, d_n)$, and (2) the sum of similarities between d_n and the other documents in $q.R'$ (i.e., $\sum_{d_j \in q.R' \setminus \{d_n\}} Sim(d_j, d_n)$). As shown in Equation 3, the computation of $TRel(q, d_n)$ is simple. Consequently, we focus on efficiently computing (2).

A straightforward way is to compute $Sim(d_n, d_i)$ for each d_i and sum them up, which is time-consuming. Nevertheless, we find that some documents in $q.R$ may share the same term. To improve the computation efficiency, we propose the concept of *aggregated term weight* and we build the *aggregated term weight summaries* to store the aggregated term weight of each term appearing in a set of documents. Such summary file allows us to get the sum of similarities without computing the similarity of every document in $q.R$ with d_n.

The concept of aggregated term weight is defined as follows.

Definition 7: Aggregated Term Weight. Let w_i be a term, $d_j.v_d.w_i$ be the term frequency of w_i in $d_j.v_d$, and S be a set of documents. Denoted by $\mathsf{AW}(w_i, S)$ the aggregated term weight of w_i w.r.t. S is computed by:

$$\mathsf{AW}(w_i, S) = \sum_{d_j \in (S \setminus \{d_e\}) \wedge w_i \in d_j.v_d} \frac{d_j.v_d.w_i}{\| d_j.v_d \|}. \quad (22)$$

□

Based on Definition 7, we are able to compute the sum of similarities between d_n and the documents in S by Lemma 6.

Lemma 6:

$$\sum_{d_j \in S \setminus \{d_e\}} Sim(d_j, d_n) = \sum_{w_i \in d_n.v_d} \frac{\mathsf{AW}(w_i, S) \times d_n.v_d.w_i}{\parallel d_n.v_d \parallel} \quad (23)$$

Proof Sketch: The proof can be found in the Appendix. □

Note that the aggregated term weight summaries will be used for handling each new document d_n. However, we observe that maintaining the aggregated term summaries for each query will induce extra space overheads though it will speed up the computation of similarity. To balance the trade-off between querying efficiency and space overheads, we maintain two result sets for q, namely $q.R_1$ and $q.R_2$. The aggregated term weight summaries are built just based on the documents in $q.R_1$ (except $q.d_e$). We develop an approach to determine whether d_n belongs to $q.R_1$ or $q.R_2$, which will be discussed in Section 7.1.

6.2 Query Result Index

We proceed to introduce the query result index for storing the result of each query.

As mentioned in Section 4.3, we index the results of each query with a query result index. Given a query q, the query result index of q consists of three components: (1) Query result table of q w.r.t. $q.R_1$, which indexes the information of documents in $q.R_1$; (2) Query result table of q w.r.t. $q.R_2$, which indexes the information of documents in $q.R_2$; (3) Query result summaries for $q.R_1 \setminus \{q.d_e\}$, which stores the aggregated term weight of each term of documents in $q.R_1 \setminus \{q.d_e\}$.

Id	$TRel$	Accumulated Similarity
$d_z.id$	$TRel(q_i, d_z)$	0
$d_y.id$	$TRel(q_i, d_y)$	$Sim(d_y, d_z)$
$d_x.id$	$TRel(q_i, d_x)$	$Sim(d_x, d_y) + Sim(d_x, d_z)$
$d_w.id$	$TRel(q_i, d_w)$	$Sim(d_w, d_x) + Sim(d_w, d_y) + Sim(d_w, d_z)$
...	...	...
$d_h.id$	$TRel(q_i, d_h)$	...
$d_e.id$	$TRel(q_i, d_e)$	$Sim(d_e, d_h) + \cdots + Sim(d_e, d_z)$

Table 3: Query Result Table of q_i w.r.t. $q_i.R_1$

Term	Aggregated Term Weight
w_i	$\mathsf{AW}(w_i, S_d)$
w_j	$\mathsf{AW}(w_j, S_d)$
w_k	$\mathsf{AW}(w_k, S_d)$
...	...
w_r	$\mathsf{AW}(w_r, S_d)$

Table 4: Query Result Summaries for $q_i.R_1 \setminus \{q_i.d_e\}$

Table 3 illustrates the query result table of q_i w.r.t. $q_i.R_1$. The documents in $q_i.R_1$ are sorted in descending order of their arrival times. In particular, for each document d_i we store: (1) Document id; (2) Text relevance between q_i and d_i; (3) Accumulated similarities of d_i, computed by Equation 24, is the sum of similarities between d_i and the documents in $q_i.R$ with arrival times earlier than d_i.

$$Sim_{acc}(q_i.R, d_i) = \sum_{d_j \in q_i.R \wedge d_j.t_c > d_i.t_c} Sim(d_i, d_j). \quad (24)$$

Based on the query result tables for $q_i.R_1$ and $q_i.R_2$, $\mathsf{dr}_{q_i}(q_i.d_e)$ can be computed as follows:

$$\mathsf{dr}_{q_i}(q_i.d_e) = \alpha \times TRel(q_i, q_i.d_e) + \frac{2-2\alpha}{k-1}(k - 1 - Sim_{acc}(q_i.R, q_i.d_e)). \quad (25)$$

Table 4 illustrates the query result summaries of $q_i.R_1 \setminus \{q_i.d_e\}$.

Individual Filtering Steps: We present the procedures for processing a new document d_n over a query q. When d_n arrives, we first visit the query result table that contains d_e to retrieve $TRel(q, q.d_e)$ and $Sim_{acc}(q.R, q.d_e)$. Then we compute $\mathsf{dr}_q(q.d_e)$ based on Equation 25. Next we visit the query result summaries to retrieve $\mathsf{AW}(w_i, q.R_1)$ for each $w_i \in q.\psi$ and visit the document lists to retrieve the text information of each $d_i \in q_i.R_2$. Finally we compute $\mathsf{dr}_q(d_n)$ and compare it with $\mathsf{dr}_q(q.d_e)$ to determine whether d_n can be a result of q. If d_n is a result of q, we update the two query result tables and the query result summaries.

7. ALGORITHM FOR DAS QUERY

In this Section, we first present our techniques for updating the indexing structure (Section 7.1), and then we introduce the algorithm for processing DAS queries when a new document arrives (Section 7.2).

7.1 Index Update

We observe that if a query result set $q.R$ is updated by a new document d_n, the oldest document in $q.R$ is discarded, and the second oldest document in $q.R$ will become the oldest document in the new result set $q.R'$. Consequently, we need update the following four components: (1) Query result table of q; (2) Aggregated term weight summaries of q; (3) The MCSs in the group-based query result summaries of the blocks that q belongs to. The update of query result table is quite straightforward. We proceed to introduce the updates of aggregated term weight summaries and the MCSs.

Update of Aggregated Term Weight Summaries. Recall that if a new document d_n is a result of q, we need determine whether the term weights of d_n should be included by the aggregated term weight summaries. In other words, we need determine whether d_n belongs to $q.R_1$ or $q.R_2$. We develop an approach to optimizing querying efficiency while taking into account the available memory space. Specifically, we first introduce a system parameter Φ_{max} denoting the total available memory space for aggregated term weight summaries and maintain two document sets, namely $q.R_1$ and $q.R_2$. The aggregated term weight summaries are built just based on the documents in $q.R_1$ (except $q.d_e$). If storing the aggregated term weights of d_n will use out the available memory, we move d_n into $q.R_2$ and do not generate aggregated term weights for d_n; otherwise we put d_n into $q.R_1$ and update the aggregated term weight summaries by the term weights of d_n.

Update of Group-based Query Result Summaries. For every new document d_n and every evaluated block b_i we need update the MCSs of b_i that are affected by the results changes of queries in b_i induced by d_n. However, such frequent updating operations are time-consuming. To alleviate the burden of updating $\mathfrak{S}_w(b_i)$, we do not update $\mathfrak{S}_w(b_i)$ each time when the result of a query in b_i is changed. Instead, when we reach the end of each block b_i we just remove the MCSs in $\mathfrak{S}_w(b_i)$ of which documents are affected by the results changes induced by d_n. Note that such updating method does not affect the correctness of the group filtering condition generated from b_i and the next new document. We can easily prove it based on Equation 19. If the current number of MCSs in $\mathfrak{S}_w(b_i)$ divided by the initial number of MCSs in $\mathfrak{S}_w(b_i)$ generated by Algorithm 1 is smaller than the system parameter δ_s ($\delta_s \in [0, 1]$), we

invoke Algorithm 1 for re-generating $\mathfrak{S}_w(b_i)$. Note that according to Equation 19 the generated group filtering condition is still correct.

7.2 Document Processing

Recall that for each block b_i from the postings lists of terms in d_n, we compute $\widetilde{\mathsf{TRel}}_{max}(b_i, d_n)$ by Equation 18 and compute $\widetilde{\mathsf{Sim}}_{min}(b_i, d_n)$ based on Equation 19.

Then we check whether b_i can be filtered without evaluating each individual query in b_i based on Lemma 7.

Lemma 7: d_n can be discarded by the queries in b if:

$$\alpha \times \widetilde{\mathsf{TRel}}_{max}(b, d_n) + \frac{2-2\alpha}{k-1}(k-1-\widetilde{\mathsf{Sim}}_{min}(b, d_n)) \leq \widetilde{\mathsf{FT}}_b$$

Proof Sketch: Based on Equation 11, Lemma 2, and Lemma 3, we can easily derive the conclusion. □

Lemma 7 enables us to efficiently determine whether d_n cannot be a result for any query in a block. If it can be determined, we move the cursor to the first query in the next block; otherwise the cursor is forwarded to the next query, and we need activate individual filtering steps for checking each individual query in the block to determine whether d_n is a result.

We are now ready to present our algorithm for maintaining the top-k results of individual queries over each new document d_n in a stream. The algorithm traverses the postings lists of every term in d_n concurrently, with Lemma 7 deployed to prune the search space.

Algorithm 2: DocumentProcess (**Document** d_n)

```
1  Result ← ∅; S_w ← d_n.ψ;
2  for each w_i ∈ S_w do
3  |  p_{w_i} ← id of the first query in I(w_i);
4  while S_w ≠ ∅ do
5  |  w_m ← term with the minimum p_{w_i} for all w_i ∈ S_w;
6  |  p_{w_m} ← FindNext(I(w_m), {p_{w_i}}_{w_i∈S_w}, Result);
7  |  if p_{w_m} reaches the end of the current block b_c then
8  |  |  Update DTRel_min(b_c) and max{TRel(q_i, d_e)|q_i ∈ b_c};
9  |  |  Q_u ← {q_j|q_j ∈ b_c ∧ q_j ∈ Result};
10 |  |  for each q_j ∈ Q_u do
11 |  |  |  Remove the MCS of b_c containing q_j.d_e;
12 |  |  if |S_{w_m}(b_c)^{cur}|/|S_{w_m}(b_c)^{initial}| <  δ_s then
13 |  |  |  GreedyMcsGen (b_c, U_{w_m}(b_c),
   |  |  |  {Q_s(b, d_i)|d_i ∈ U_{w_m}(b_c)});
14 |  if p^{w_m} reaches the end of I(w_m) then
15 |  |  S_w ← S_w \ w_m;
```

Algorithm 2 shows the pseudo code. We first introduce the variables in Algorithm 2. S_w is for storing the terms in $d_n.v_d$ of which postings lists are not completely traversed, $|\mathfrak{S}_w(b_c)^{cur}|$ denotes the cardinality of current $\mathfrak{S}_w(b_c)$, and $|\mathfrak{S}_w(b_c)^{initial}|$ refers to the cardinality of initial $\mathfrak{S}_w(b_c)$.

We initialize the cursor p_{w_i} of each postings list to be its first element (line 1). Here $I(w)$ represents the postings list for term w (lines 2–3). Next, we choose the postings list of w_m whose current posting has the minimum query id among all the postings lists of the terms of d_n (line 5). Then we invoke function FindNext to evaluate the queries sequentially until w_m is not the term with the minimum p_{w_i} (line 6). If the cursor reaches the end of the current block b_c, we update $DTRel_{min}(b_c)$ and $max\{TRel(q_i, q_i.d_e)|q_i \in b_c\}$ (line 8). Then, for each query q_j of which result is updated by d_n, we remove the MCS that contains the updated $q_j.d_e$ in the result of q_j from the query result summaries of b_c (lines 9–11). If the cardinality of current $\mathfrak{S}_{w_m}(b_c)$ divided by the cardinality of the initial $\mathfrak{S}_{w_m}(b_c)$ is smaller than δ_s, which is a system parameter, then we invoke GreedyMcsGen to re-generate the $\mathfrak{S}_{w_m}(b_c)$ based on the current results of queries in b_c (lines 12–13). We will remove w_m from S_w if p_{w_m} reaches the end of $I(w_m)$ (lines 14–15). The above process is repeated until S_w is empty.

Function FindNext($I(w)$, $\{p_w\}_{w\in S_w}$, $Result$)

```
1  p_c ← p_w;
2  b_c ← the block containing p_c;
3  if p_c = b_c.first then
4  |  if Lemma 7 is satisfied then
5  |  |  b_c ← b_c.next; p_c ← b_c.first;
6  else
7  |  q ← the query indicated by p_c;
8  |  Invoke individual filtering steps;
9  |  if d_n is the result of q then
10 |  |  Result.add(q);
11 |  |  Update the the query result table and aggregated term weight
   |  |  summaries of q;
12 |  p_c ← p_c.next;
13 return p_c;
```

Function FindNext has three parameters. $I(w)$ denotes the postings list we are evaluating now, $\{p_w\}_{w\in S_w}$ denotes a set of cursors w.r.t. the terms in S_w, and $Result$ is the set of queries of which results are updated by d_n. First, p_c and b_c are respectively initialized as the current posting and block (lines 1–2). If p_c is the first posting in b_c, i.e., none of the queries in b_c have been evaluated, then we check whether b_c can be skipped based on Lemma 7. Specifically, if Lemma 7 is satisfied, none of the queries in b_c can include d_n as a result, and thus we skip b_c and forward the cursor to the next block (lines 4–6). However, if p_c is not the first posting in b_c, which indicates that b_c cannot be skipped as a whole, then we invoke the individual filtering steps introduced in Section 6.2 for determining whether d_n can be a result of q (line 8). If d_n is a result of q, we add q as the query that matches d_n (line 10). Then we update the the query result table and aggregated term weight summaries of q (line 11). Finally, we move p_c to the next posting and return p_c.

8. EXPERIMENTAL STUDY

8.1 Baselines

Because no algorithm exists for processing DAS queries, we develop two baselines by utilizing existing index structures for processing a large number of DAS queries, namely IRT and BIRT. In addition, to study the effectiveness of the DAS query, we present how to extend two existing diversity-aware algorithms, namely DisC [12] and MSInc [27], to process subscription (standing) queries over the text stream. Due to the space limitation, we present the baselines in Appendix A.

8.2 Experimental Setup

Our experiments are conducted on a real-life dataset collected from Twitter, which contains 10 million tweets collected from Mar 2012 to Nov 2012. There are 2.4 million distinct terms in the dataset (excluding stop-words), and the average number of terms in each tweet is 8.

Query Generation: We generate two set of queries for our experiments. The first set of queries, denoted by LQD, consists of 2M subscription queries. To generate a query for LQD, we first randomly pick a tweet in the dataset, then we randomly choose a specified number of terms (1 to 5) from the tweet as the query keywords (i.e.,$q.\psi$). Note that the tweets posted by the user may reveal

the interests of the user, and thus the subscription query generated in this way would be close to real queries. Actually, popular words or trending topic words are more likely to appear in the queries generated in this way than less popular words.

We also generate a smaller set of queries (denoted by SQD) that contains 100K subscription queries. For each query in SQD we randomly choose 1-5 trending topics as query keywords from the Twitter 2012 trending topic page[2]. These queries might be more similar to the real-life search queries.

We use LQD, the larger set of queries, to study the runtime performance of our proposed methods for processing the DAS queries, and use SQD to evaluate the effectiveness of the DAS query by comparing with DisC and MSInc. We also conduct a set of experiments to compare the efficiency of our system and the two other diversity-aware systems, DisC and MSInc.

Default values for parameters are presented in Table 5. Note that due to the space limitation, we put some experimental results in Appendix B.

Table 5: Default Values for each Parameter

Parameter	Setting	Default
# query terms	1 to 15	1 to 5
# document terms	5 to 20	N.A.
parameter α	0.1 to 0.9	0.3
number of results	10 to 50	30
# postings in each block	32 to 4096	256
decaying scale	0.1 to 0.9	0.5
Φ_{max}	0.5GB to 2GB	0.5GB
δ_s	0.1 to 0.9	0.5
# queries	2M to 8M	2M
sliding window size $\|W_f\|$	5K to 20K	10K

We implemented all algorithms in Java on a PC with Intel(R) Core(TM) i7-4770k @3.50GHz and 16GB RAM.

8.3 Experimental Results for DAS Queries

We evaluate the runtime performance of document processing for each method developed for processing DAS queries on LQD. For a new document d_n, the runtime cost for processing d_n consists of the following two parts: (1) Time cost for finding the queries that can include d_n as their results; (2) Time cost for index update, which includes the cost for updating query result summaries, query result tables, and aggregated term weight summaries.

We use GIFilter to denote the method with both group filtering technique and individual filtering technique, and we use IFilter to denote the method with only individual filtering technique applied.

Time Effect: In this set of experiments, each method runs for 120 minutes (which is simulation duration, denoted by Δt_{sim}), on LQD. We set the decaying scale $B^{-\Delta t_{sim}}$ at 0.5. To make sure that all the methods can handle, we issue 1 document and 1 new query each second. At the beginning, each method is initialized with 2,000,000 DAS queries. We report the average runtime of document processing and the average runtime for query insertion during each period of 10 minutes.

Figure 4(a) shows that our GIFilter exhibits the best performance on the larger queryset LQD. It is able to reduce the runtime of the best baseline BIRT by 60%-70% . The reasons could be explained as follows. For IRT, we need to check each posting in the postings list of each term of a new document. While for BIRT, postings are indexed by blocks, which may help prune the queries in a block-based manner during the search of postings list. Therefore, BIRT performs slightly better than IRT. However, neither IRT nor BIRT have the aggregated term weight summaries for each query. They have to evaluate every document in the results of the queries in the postings lists that are not skipped. Consequently, with the help of aggregated term weight summaries IFilter is able to reduces the runtime of BIRT by 25% to 30%. Nevertheless, the pruning technique of IFilter is does not consider the diversity aspect w.r.t. new document and a block of queries. We find that GIFilter reduces the runtime of IFilter by 40% to 50%. Such performance improvement is contributed by the minimal covering sets in the query result summaries for each block.

Figure 4(b) shows the performance of query insertion for each method. Because IFilter maintains for each query aggregated term weight summaries, and GIFilter maintains for each block query result summaries (apart from the aggregated term weight summaries for each query). The query insertion for IFilter and GIFilter takes longer than that of the other methods. However, the time for the query insertion is negligible compared with the time for document processing. Furthermore, in the publish/subscribe scenario the frequency of query insertion is normally much lower than that of document arrivals, and thus the portion of query insertion cost will be even smaller than that shown in Figure 4(b). Hence, for some experiments we only show the document processing cost while ignoring the query insertion cost.

Effect of the Number of Query Keywords: We proceed to evaluate the effect of the number of query keywords on efficiency. Figure 5(a) shows that the runtime of document processing for all the methods increases as we increase $|q.\psi|$, but the increase is mild. This is because that increasing the number of query keywords will increase the average number of postings of each postings list, which will lead to an increase in the number of queries to be evaluated for each document. We also observe that GIFilter consistently improves on the runtime performance of BIRT by at least 50%.

Figure 5(b) demonstrates the runtime of query insertion for each method. We observe that the runtime of all methods increases slightly as $|q.\psi|$ increases.

Number of Maintained Query Results: This experiment evaluates the effect of parameter k on the performance. Figure 6 shows that the runtime of all methods for document processing moderately increases as we increase the number of results maintained by each query. The reason is that the higher value of k will increase the cost for computing the diversity score.

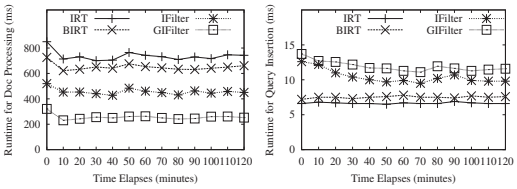
Number of Indexed Queries: This experiment is to evaluate the effect of the number of indexed queries. The number of queries scales from 2M to 8M. Obviously, increasing the number of indexed queries leads to the increase of postings in each postings list. Hence, more postings will be retrieved and evaluated while processing a new document. Figure 7(a) and Figure 8 show that both the runtime for document processing and the index size exhibit a linearly increasing trend for all methods as we increase the number of indexed queries. Figure 7(b) demonstrates the performance of query insertion as we increase the number of indexed queries. We notice that the number of indexed queries does not have a significant impact on the performance of query insertion.

8.4 Comparison with Other Diversity-Aware Systems

8.4.1 Effectiveness

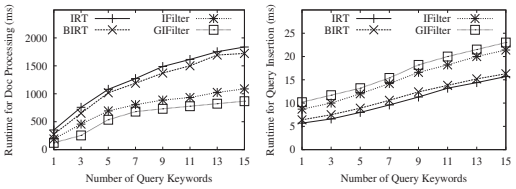
To evaluate the result quality of our DAS query, we conduct a user study on the users' satisfaction on the result sets produced by each method (i.e., DisC, MSInc, and GIFilter). Note that IRT, BIRT, IFilter, and GIFilter are all developed for processing DAS

[2]https://2012.twitter.com/en/trends.html



(a) Doc Processing (b) Query Insertion

Figure 4: Time Effect on LQD



(a) Doc Processing (b) Query Insertion

Figure 5: Number of Query Keywords

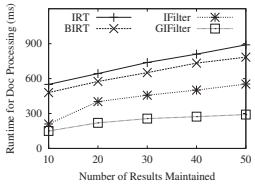
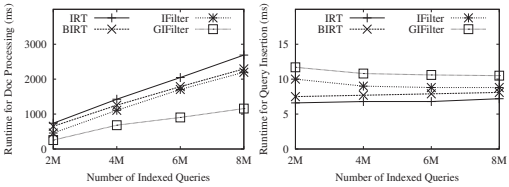


Figure 6: Effect of k



(a) Doc Processing (b) Query Insertion

Figure 7: Scalability on LQD

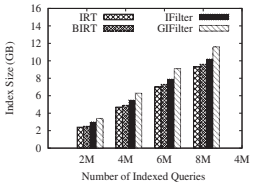
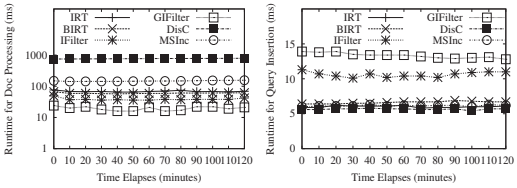


Figure 8: Memory Cost



(a) Doc Processing (b) Query Insertion

Figure 9: Time Effect on SQD

queries, and they produce the same result. Hence, we only use GIFilter for user study. We follow the methodology and procedure of user study in previous work [40] on diversity-aware system, which is detailed as follows.

First, we generate 50 subscription queries by choosing 50 trending topics as query keywords from the 2012 Twitter trending topics.

Second, we process the 50 subscription queries over the tweets stream for each method under different parameter settings. For MSInc and GIFilter we set the parameter α that specifies the trade-off between relevance and diversity to be 0.3 and 0.7, and the number of results (i.e., k) to be 5. For DisC, we set the size of each window frame (i.e., $|W_f|$) to be 10K. Because DisC does not offer an parameter to directly set the number of results returned, we fine-tune the similarity threshold r such that the queries return 5 results on average. We record 3 result sets of each query at 3 randomly selected timestamps such that at each selected timestamp, the set of results returned by DisC contains exactly 5 results to avoid the rating bias induced by the number of results.

Third, for each result set, we display the query keywords, the timestamp where the result set is recorded, and the content along with the arrival time of each tweet in the set. We ask 3 annotators to rate the quality of the result set and record the average score on each rating aspect. Specifically, each user is required to rate his/her satisfaction towards the result set in terms of the following three aspects: (1) The relevance of each result (scales ranged from "not relevant" to "very relevant", mapped to values 1 to 5); (2) The recency of each result (scales ranged from "not fresh/timely" to "very fresh/timely", mapped to values 1 to 5); (3) Whether the result set reflects a broad or narrow range of his/her interests (scales ranged from "very narrow" to "very broad", mapped to values 1 to 5); (4) Overall satisfaction with the results (scales ranged from "not satisfied" to "very satisfied", mapped to values 1 to 5).

Table 6: User Study Result

Method	Relevance	Recency	Range of Int.	Overall
GIFilter α=0.3	3.5	4.2	4.0	3.9
GIFilter α=0.7	4.0	4.5	2.9	3.2
MSInc α=0.3	3.4	4.0	4.1	3.9
MSInc α=0.7	4.2	4.4	3.0	3.3
DisC	2.6	3.6	3.5	3.1

Table 6 shows the result of our user study. Following [19], we compute the consistency of different annotators by computing the average linearly weighted Cohen's kappa between all pairs of raters for each rating aspect of each method. The kappa values for relevance are in the range of 0.52-0.67, for recency in the range of 0.61-0.78, for range of interests in the range of 0.36-0.55, and for overall satisfaction in the range of 0.38-0.42. We find that the results returned by GIFilter and MSInc have similar quality in terms of all rating aspects. In particular, when α is set at 0.3 (i.e., diversity is playing a more important role in query processing), GIFilter and MSInc produce the result sets with the same quality w.r.t. overall satisfaction. When α is set at 0.7 (i.e., relevance and recency weight more in query processing), the result quality of MSInc is slightly better than that of GIFilter. In addition, the result quality of DisC is substantially worse than that of GIFilter and MSInc.

8.4.2 Efficiency

In this set of experiments, we study the efficiency of DisC and MSInc in comparison to our proposed methods for processing DAS queries. Because DisC and MSInc fail to process the queries in LQD over the text stream with 1 document and 1 new query each second (i.e., the time of processing 1 document over LQD exceeds 1 second), we compare with DisC and MSInc on SQD.

Similar to the settings for LQD, each method runs for $\Delta t_{sim} = 120$ minutes and the decaying scale $B^{-\Delta t_{sim}}$ is set at 0.5. To make sure that all the methods can handle, we also issue 1 document and 1 new query each second. Because the subscription queries in DisC are executed periodically over a sliding window, we execute each query in DisC every 10 minutes. To ensure the comparability of document processing between DisC and the other streaming-based approaches, we measure the runtime of document processing for DisC by the following equation.

$$DocProcessTime_{DisC} = \frac{TotalQueryingTime(\Delta t)}{DocNum(\Delta t)},$$

where $TotalQueryingTime(\Delta t)$ denotes the total time spent for query processing during the time period of Δt, and $DocNum(\Delta t)$ denotes the number of documents arrived during Δt. At the beginning, each method is initialized with 100,000 queries. We report the average runtime of document processing and the average runtime for query insertion during each period of 10 minutes.

Figure 9(a) demonstrates that our proposed methods for processing DAS queries outperform DisC and MSInc on SQD. In particular, GIFilter is able to reduce the runtime of DisC and MSInc by more than an order of magnitude.

Figure 9(b) shows the performance of query insertion on SQD. Similar to the result on LQD (i.e., Figure 4(b)), the query insertion for IFilter and GIFilter takes longer than that of the other methods on SQD. The detailed reasons have been presented in Section 8.3. Nevertheless, the time for the query insertion is negligible compared with the time for document processing.

9. RELATED WORK

Streaming-based Diversification. Our problem is related to the problem of continuously monitoring k-most diverse items over continuous data. Drosou et al. [13] propose a cover-tree based index for continuously maintaining the k-most diverse set over a sliding window based on the Max-min diversification. Minack et al. [27] propose an approach to find a diverse set by processing items in a streaming manner based on Max-sum or Max-min diversification. Borodin et al. [5] study the Max-sum diversification problem on stream data and they adopt monotone submodular functions for measuring the diversity. Panigrahi et al. [28] model the problem of selecting a diverse set from a stream of items as a covering problem. Specifically, each item is annotated by a set of features, and the problem is to select a diverse set to cover as many features as possible. However, these solutions focus on a single query and cannot be used to efficiently maintain a large number of result sets simultaneously, which is a major challenge for top-k publish/subscribe systems [32]. In addition, the methods proposed in [5, 13] are not for a text stream.

Cheng et al. [7] study the problem of selecting the minimum representative (diverse) subset of tweets for a small group of queries from the same user. The problem is substantially different from our problem in three aspects. First, it returns a single set of tweets as the result for all queries in the group while we maintain a set of results for each query. Second, their diversity model is different from our work. Instead of using inter-tweet similarity metric, they assign each tweet with a value on the selected diversity dimension (e.g., timestamp or sentiment polarity). Third, their proposal does not handle millions of subscription queries as we do.

Lourenco et al. [21] study the problem of selecting a set of training items periodically over the text stream based on two criteria, namely adaptiveness and memorability, for sentiment analysis. Their problem is different from our problem, and we do not see their techniques can be used for our problem.

Top-k Publish/subscribe over Text Stream. Closest to our problem setting is the existing work on top-k publish/subscribe systems [6, 17, 18, 29, 32], in which published items trigger a subscription only if it ranks among the top-k published items for the subscription. In the setting of most of these systems [17, 18, 29], the relevance of an item remains constant during a pre-specified time interval, and once its lifetime exceeds, the item simply expires. The expired item is then replaced by the most relevant unexpired item. The setting is different from our setting where time is part of the ranking score. In the sense, the setting of top-k publish/subscribe systems in [6, 32] is similar to ours. For [32] the published items are tweets and the subscriptions are news. While for [6] the published items are geo-tagged tweets and the subscriptions are Points of Interest (POIs). In the above two studies, the published items do not have a fixed expiration time. Instead, time is a part of the relevance score, which decays as time passes. Older items retire from the top-k only when new items that score higher arrive. The inverted files are used as the indexes and the classic information retrieval methods are adapted for the ranking. Our work differs from these studies in that we consider the diversity of published items for each subscription as part of the ranking score, which renders the solutions [6, 32] inapplicable, and this also introduces new challenges for top-k publish/subscribe.

In addition, Rao et al. [30] study the problem of processing continuous top-k queries over document streams. They propose a graph based query indexing structure based on the "covering relationship" among queries. However, it does not consider the diversity issue. We do not see a way to adapt it for handling DAS queries.

Query Result Diversification. Query result diversification has been extensively studied for recent years. Most previous work on query result diversification can be classified into the following two categories: implicit and explicit [31, 39]. Our problem belongs to the former category.

The implicit approaches assume that similar documents cover similar aspects and model inter-document dependencies (e.g., Max-sum diversification and Max-min diversification). Most work on implicit query result diversification aims at finding a set of k items based on a scoring function that considers both relevance and diversity (e.g., [36]). Khan et al. [22] study the problem of the concurrent diversification problem for answering multiple diversified top-k queries over static data. Angel et al. [3] study the problem of diversified keyword search in documents based on user behavior model. Fraternali et al. [15] aim at answering diversified top-k query over low-dimensional vector space. Zhang et al. [38] focus on the spatial keyword search diversification on road networks. Abbar et al. [1] study the problem of set-based recommendation of diverse articles. Diversity in [1] is measured through incorporating entities and sentiment extracted from comments of articles. Recently, Liang et al. [25] tackle the problem of result diversification by data fusion approaches. However, these problems substantially differ from our diversity-aware publish/subcribe problem and their proposals are not for stream data.

The explicit approaches for query result diversification model a set of query topics (aspects) and return documents for each of them (e.g., [8, 9, 31, 33, 35]). However, the query aspects are unavailable for our problem.

10. CONCLUSIONS

We consider the problem of maintaining up-to-date results for a large number of DAS queries. The DAS query takes into account text relevance, document recency, and result diversity in evaluating the query rseult. We propose a novel mechanism to efficiently processing a large number of DAS queries. In particular, based on the concept of filtering conditions, we develop group filtering technique and individual filtering technique for determining whether a new document can be a result of each DAS query. The experimental results on real-world dataset show that our solution is able to achieve a reduction of the processing time by 60–75% compared with two baselines.

11. ACKNOWLEDGEMENT

This work is supported by a Singapore MOE AcRF Tier 2 Grant (ARC30/12) and a grant awarded by Microsoft Research.

12. REFERENCES

[1] S. Abbar, S. Amer-Yahia, P. Indyk, and S. Mahabadi. Real-time recommendation of diverse related articles. In *WWW*, pages 1–12, 2013.

[2] G. Amati, G. Amodeo, and C. Gaibisso. Survival analysis for freshness in microblogging search. In *CIKM*, pages 2483–2486. ACM, 2012.

[3] A. Angel and N. Koudas. Efficient diversity-aware search. In *SIGMOD*, pages 781–792, 2011.

[4] T. Apaydin and H. Ferhatosmanoglu. Access structures for angular similarity queries. *IEEE Trans. Knowl. Data Eng.*, 18(11):1512–1525, 2006.

[5] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *PODS*, pages 155–166, 2012.

[6] L. Chen, G. Cong, X. Cao, and K.-L. Tan. Temporal spatial-keyword top-k publish/subscribe. In *ICDE*, 2015.

[7] S. Cheng, A. Arvanitis, M. Chrobak, and V. Hristidis. Multi-query diversification in microblogging posts. In *EDBT*, pages 133–144, 2014.

[8] V. Dang and W. B. Croft. Diversity by proportionality: an election-based approach to search result diversification. In *SIGIR*, pages 65–74, 2012.

[9] V. Dang and W. B. Croft. Term level search result diversification. In *SIGIR*, pages 603–612, 2013.

[10] Y. Diao, P. M. Fischer, M. J. Franklin, and R. To. Yfilter: Efficient and scalable filtering of XML documents. In *ICDE*, pages 341–342, 2002.

[11] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Record*, 39(1):41–47, 2010.

[12] M. Drosou and E. Pitoura. Disc diversity: result diversification based on dissimilarity and coverage. *PVLDB*, 6(1):13–24, 2012.

[13] M. Drosou and E. Pitoura. Dynamic diversification of continuous data. In *EDBT*, pages 216–227, 2012.

[14] M. Efron and G. Golovchinsky. Estimation methods for ranking recent information. In *SIGIR*, pages 495–504. ACM, 2011.

[15] P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Top-k bounded diversification. In *SIGMOD*, pages 421–432, 2012.

[16] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009.

[17] P. Haghani, S. Michel, and K. Aberer. Evaluating top-k queries over incomplete data streams. In *CIKM*, pages 877–886, 2009.

[18] P. Haghani, S. Michel, and K. Aberer. The gist of everything new: Personalized top-k processing over web 2.0 streams. In *CIKM*, pages 489–498, 2010.

[19] C. Hauff, F. de Jong, D. Kelly, and L. Azzopardi. Query quality: user ratings and system predictions. In *SIGIR*, pages 743–744, 2010.

[20] C. A. J. Hurkens and A. Schrijver. On the size of systems of sets every t of which have an sdr, with an application to the worst-case ratio of heuristics for packing problems. *SIAM J. Discrete Math.*, 2(1):68–72, 1989.

[21] R. L. Jr., A. Veloso, A. M. Pereira, W. M. Jr., R. Ferreira, and S. Parthasarathy. Economically-efficient sentiment stream analysis. In *SIGIR 2014*, pages 637–646, 2014.

[22] H. A. Khan, M. Drosou, and M. A. Sharaf. Scalable diversification of multiple search results. In *CIKM*, pages 775–780, 2013.

[23] X. Li and W. B. Croft. Time-based language models. In *CIKM*, pages 469–475. ACM, 2003.

[24] H. Liang, Y. Xu, D. Tjondronegoro, and P. Christen. Time-aware topic recommendation based on micro-blogs. In *CIKM*, pages 1657–1661, 2012.

[25] S. Liang, Z. Ren, and M. de Rijke. Fusion helps diversification. In *SIGIR*, pages 303–312, 2014.

[26] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[27] E. Minack, W. Siberski, and W. Nejdl. Incremental diversification for very large sets: a streaming-based approach. In *SIGIR*, pages 585–594, 2011.

[28] D. Panigrahi, A. D. Sarma, G. Aggarwal, and A. Tomkins. Online selection of diverse results. In *WSDM*, pages 263–272, 2012.

[29] K. Pripužić, I. P. Žarko, and K. Aberer. Top-k/w publish/subscribe: Finding k most relevant publications in sliding time window w. In *DEBS*, pages 127–138, 2008.

[30] W. Rao, L. Chen, S. Chen, and S. Tarkoma. Evaluating continuous top-k queries over document streams. *World Wide Web*, 17(1):59–83, 2014.

[31] R. L. T. Santos, C. Macdonald, and I. Ounis. Exploiting query reformulations for web search result diversification. In *WWW*, pages 881–890, 2010.

[32] A. Shraer, M. Gurevich, M. Fontoura, and V. Josifovski. Top-k publish-subscribe for social annotation of news. *PVLDB*, 6(6):385–396, 2013.

[33] I. Szpektor, Y. Maarek, and D. Pelleg. When relevance is not enough: promoting diversity and freshnessin personalized question recommendation. In *WWW*, pages 1249–1260, 2013.

[34] J. Teevan, D. Ramage, and M. R. Morris. #twittersearch: a comparison of microblog search and web search. In *WSDM*, pages 35–44, 2011.

[35] S. Vargas, P. Castells, and D. Vallet. Explicit relevance models in intent-oriented information retrieval diversification. In *SIGIR*, pages 75–84, 2012.

[36] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. T. Jr., and V. J. Tsotras. On query result diversification. In *ICDE*, pages 1163–1174, 2011.

[37] L. Wu, W. Lin, X. Xiao, and Y. Xu. LSII: an indexing structure for exact real-time search on microblogs. In *ICDE*, pages 482–493, 2013.

[38] C. Zhang, Y. Zhang, W. Zhang, X. Lin, M. A. Cheema, and X. Wang. Diversified spatial keyword search on road networks. In *EDBT*, pages 367–378, 2014.

[39] Y. Zhu, Y. Lan, J. Guo, X. Cheng, and S. Niu. Learning for search result diversification. In *SIGIR*, pages 293–302, 2014.

[40] C. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, pages 22–32, 2005.

APPENDIX

A. BASELINES

A.1 Baselines for Processing DAS Queries

Inverted File Plus Query Result Tables (IRT): This baseline is developed for processing DAS queries. It consists of a query inverted file and query result tables, which is referred to as IRT. We use the inverted file to index DAS queries. We also maintain a query result table for each query. When a new document d_n arrives, we traverse the postings lists for each term in $d_n.v_d$ in the Document-at-a-Time (DAAT) [26] manner. For each posting (query) q, we visit its corresponding query result table and determine whether d_n can update the result of q based on Definition 3. In addition, for improving the filtering efficiency, we first compute $\alpha \times TRel(q, d_n) + 2 \times (1 - \alpha)$, which is based on Equation 7 by regarding $\mathsf{d}(d_n, d_i)$ as 1. Then we compare it with $\mathsf{dr}_q(q.d_e)$ based on the individual filtering condition in Definition 3. If $\alpha \times TRel(q, d_n) + 2 \times (1-\alpha) \leq \mathsf{dr}_q(q.d_e)$, we skip q since it d_n will not a result of q; otherwise, we further check each document in $q.R$ for computing an exact $\mathsf{dr}_q(d_n)$. If d_n is a result of q, we update the query result table of q. Note that this baseline utilizes the filtering threshold proposed in Definition 3, and it is not the straightforward method introduced in Section 3, which performs much worse.

Block based Inverted File Plus Query Result Tables (BIRT): This baseline uses a block based inverted file to index the DAS queries, which is referred to as BIRT. BIRT partitions each postings list into blocks, each of which contains a pre-specified number of postings. Similar to Algorithm 2, while processing each new document d_n we traverse the corresponding postings lists with the forward block skipping technique. When we visit block b_i, we use Lemma 7 to determine whether b_i can be skipped by d_n. Because BIRT does not contain the query result summaries for each block, we are cannot derive $\widetilde{\mathsf{Sim}}_{min}(b_i, d_n)$ based on the MCSs for b_i as it is done in our proposed method. Hence, we use 0 as the value of $\widetilde{\mathsf{Sim}}_{min}(b_i, d_n)$, which is the minimum possible value of similarity.

A.2 Algorithms for Processing DisC

The Dissimilar and Covering Diversity (DisC Diversity) System [12] is a state-of-the-art diversity-aware system for processing a single query over static data.

Given a set of items P, DisC [12] aims at selecting a *Dissimilar and Covering* subset $S \subseteq P$ such that: (1) All items in P are similar with at least one item in S; (2) No two items in S are similar with each other. Here two items o_1 and o_2 are considered to be similar if $dist(o_1, o_2) \leq r$, where r is a tuning parameter and $dist(o_1, o_2)$ can be any distance function. Two algorithms, BasicDisC and GreedyDisC [12], are developed for processing a single DisC query over static data, and we extend them to process standing queries over text stream as follows.

The similarity measurement of DisC must be a distance metric, and we use angular similarity to measure the similarity between two documents. Note that angular similarity has been used in information retrieval for semantic analysis of text documents [4]. Specifically, given two documents d_i and d_j, we have:

$$Sim(d_i, d_j) = 1 - \frac{cos^{-1}(CosineSimilarity(d_i, d_j))}{\pi}.$$

We assume that documents from the text stream are already indexed by an inverted file. In other words, we do not consider the cost of maintaining inverted file for the text stream, which is a setting favoring the DisC system. We maintain a sliding window of size $|W_f|$ over the text stream and move forward the window frame periodically. For each window frame, we employ BasicDisC or GreedyDisC algorithm to find a *Dissimilar and Covering* subset of documents within the window frame.

GreedyDisC produces results with better quality than BasicDisC does. But BasicDisC is much more efficient than GreedyDisC. We choose the most favorable setting for DisC, namely, we use BasicDisC for efficiency study and use GreedyDisC for effectiveness study. In the rest of this paper, we call them DisC uniformly.

A.3 MSInc Algorithm

MSInc [27] is the state-of-the-art streaming-based diversification algorithm that takes diversity, relevance, and recency into consideration. Similar to DisC, MSInc is also developed for processing a single query over a static set of items. However, it is a stream-based approach that processes items in an incremental manner for maintaining a diverse set. Hence, MSInc can be extended to process standing query over text stream.

B. ADDITIONAL EXPERIMENTS

In this section, we present some additional experimental results.

Effect of Block Size: This round of experiments is to evaluate the performance of the methods utilizing block structure, including BIRT, IFilter, and GIFilter, when we vary the the number of postings in each block. Figure 10 shows the trend of the document processing cost w.r.t. the block size. If the block size is too small, then the number of blocks we need to evaluate will increase. On the other hand, if the block size is too large, the possibility for skipping a block will decrease despite the reduction of the number of blocks to be visited while processing a new document. The performance is not significantly affected by the block size for all methods when we vary the number of postings in each block from 128 to 2048. However, when we set it below 128 or over 2048, The performance of GIFilter will deteriorate significantly.

Arrival Rate: We vary the arrival rates of both documents and queries. Figure 11(a) presents the total time costs in every 1 minute for document processing when we vary the arrival rate of documents from 0 to 200 documents/minute with 2M DAS queries indexed. We find that GIFilter is capable of processing 200 documents with 2M indexed queries while the other methods fail to handle. Figure 11(b) presents the total runtime of query insertion when we vary the arrival rate of DAS queries. Although the query insertion cost of GIFilter is moderately higher than the the other three methods, the arrival rate of query is much lower than the arrival rate of document under the publish/subscribe scenario in practice.

Effect of α: In this experiment, we investigate the effect of the system parameter α. A lower value of α indicates the greater weight for the diversity score, while a larger value of α indicates more emphasis on the text relevance. From Figure 12, we observe that for IRT and BIRT the performance of document processing is greatly affected by α. The reason is that the filtering conditions generated by them are solely based on the text relevance. When α is large and text relevance dominates the ranking, their filtering conditions work better. We also observe that GIFilter exhibits a nearly consistent trend as we vary α. The reason is that the group filtering condition generated by GIFilter is based on both text relevance and diversity score.

In particular, at α = 0.9 (text relevance has a high weight), the performances of the two baselines are even slightly better than GIFilter. This is because that the effectiveness of filtering conditions

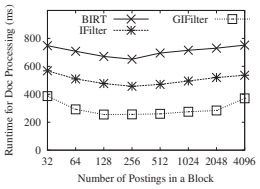
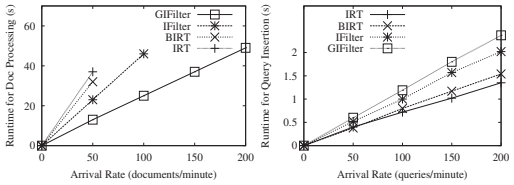


Figure 10: Block Size



(a) Doc Arr Rate (b) Query Arr Rate

Figure 11: Arrival Rate

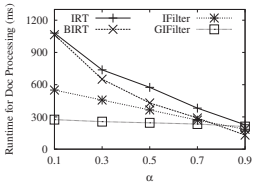


Figure 12: Effect of α

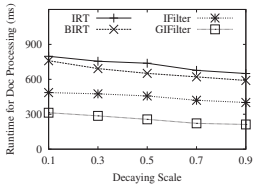


Figure 13: Decaying Scale

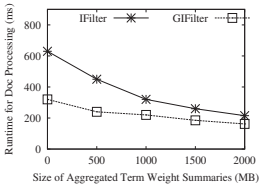


Figure 14: Effect of Φ_{max}

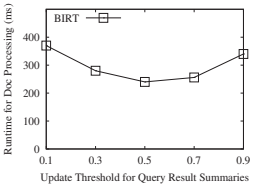


Figure 15: Effect of δ_s

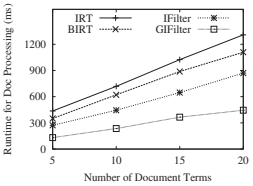


Figure 16: # Doc Terms

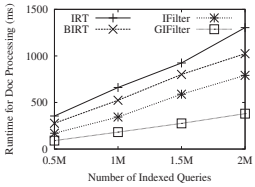


Figure 17: Scala. on SQD

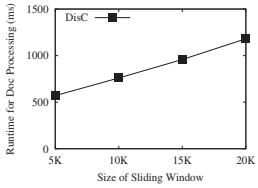


Figure 18: Effect of $|W_f|$

generated by the two baselines are close to that generated by GIFilter, and GIFilter incurs runtime overheads for maintaining query result summaries and aggregated term weight summaries.

Decaying Scale: We evaluate the effect of the decaying scale. Figure 13 shows that the runtime for document processing decreases as we increase the decaying scale. The reason is that a lower value of decaying scale will increase the number of queries that have a new document to be their results.

Effect of Φ_{max}: In this experiment, we vary the size of aggregated term weight summaries for IFilter and GIFilter. Figure 14 shows that both methods exhibit better performance as we increase the value of Φ_{max}.

Effect of δ_s: We evaluate the effect of δ_s for GIFilter. Figure 15 demonstrates that GIFilter performs best when δ_s is set as 0.5. The reason can be explained as follows. If δ_s is too small, the number of MCSs for each block may not be enough for generating an effective the group filtering condition. However, if δ_s is too large, MCSs for each block may be updated frequently, which will incur additional runtime costs.

Effect the Number of Distinct Document Terms: In this set of experiments, we vary the number of distinct terms in a document from 5 to 20. Figure 16 shows that GIFilter consistently reduces the runtime of the two baselines by 50%-70% when the number of unique document terms is varied. We also note that the document processing costs increase linearly for all methods as we increase the number of distinct terms. The reason is that the more terms there are in a new document d_n, the more postings lists are to be retrieved during the process of finding the queries that can have d_n as a result.

Scalability on SQD: We evaluate the scalability effect on SQD by varying the number of queries from 0.5M to 2M. As shown in Figure 17, when the number of queries is 2M, which is the same as the number of queries in LQD, the relative performances of IRT, BIRT, IFilter, and GIFilter are similar to those on LQD.

Effect of Sliding Window Size: We evaluate the effect of sliding window size (i.e., $|W_f|$) for DisC. Figure 18 shows that the runtime of document processing exhibits a linearly increasing trend when we vary the window size from 5K to 20K.

C. PROOFS

Proof of Lemma 1.

Based on Equation 1 and Equation 5 we have:

$$DR(q.R') - DR(q.R)$$
$$= \alpha \times \Big(\sum_{d_i \in q.R'} R(q, d_i) - \sum_{d_i \in q.R} R(q, d_i) \Big) \; +$$
$$\frac{2 \times (1-\alpha)}{k-1} \Big[\sum_{d_i \in q.R' \setminus \{d_n\}} \mathsf{d}(d_n, d_i) - \sum_{d_i \in q.R \setminus \{q.d_e\}} \mathsf{d}(q.d_e, d_i) \Big],$$

where

$$\sum_{d_i \in q.R'} R(q, d_i) - \sum_{d_i \in q.R} R(q, d_i) = R(q, d_n) - R(q, q.d_e).$$

So based on Equation 8 and Equation 7, we have Equation 9.

Proof of Lemma 2.

Let q_r be the query in G at time t_{cur} s.t.

$$\mathsf{dr}_{q_r}(q_r.d_e) = min\{\mathsf{dr}_{q_i}(q_i.d_e) | q_i \in G\}$$

Then we have:

$$min\{\mathsf{dr}_{q_i}(q_i.d_e) | q_i \in G\} =$$
$$\alpha \times TRel(q_r, d_e) + \frac{2 \times (1-\alpha)}{k-1} \sum_{d_j \in q_r.R \setminus \{q_r.d_e\}} \mathsf{d}(d_j, q_r.d_e) -$$
$$\alpha \times TRel(q_r, q_r.d_e) \times (1 - B^{-(t_{cur} - q_r.d_e.t_c)}).$$

Because $TRel(q_m, q_m.d_e) \geq TRel(q_r, q_r.d_e)$ and $q_e.d_e.t_c \leq q_r.d_e.t_c$, we can deduce that

$$TRel(q_r, q_r.d_e) \times (1 - B^{-(t_{cur} - q_r.d_e.t_c)}) \leq$$
$$TRel(q_m, q_m.d_e) \times (1 - B^{-(t_{cur} - q_e.d_e.t_c)}).$$

Then, we find that

$$DTRel_{min}(G) \leq$$
$$\alpha \times TRel(q_r, q_r.d_e) + \frac{2 \times (1-\alpha)}{k-1} \sum_{d_j \in q_r.R \setminus \{q_r.d_e\}} \mathsf{d}(d_j, q_r.d_e).$$

Thus we complete the proof.

Proof of Lemma 3.

Let $\mathsf{dr}_{q_m}(d_n) = max\{\mathsf{dr}_{q_i}(d_n)|q_i \in G\}$. According to Equation 4, we know that $T(d_n) = 1$ because $d_n.t_c = t_{cur}$, so based on Equation 7 we have:

$$\mathsf{dr}_{q_m}(d_n) = \alpha \times TRel(q_m, d_n) + \frac{2 \times (1-\alpha)}{k-1}(k - 1 - \sum_{d_i \in q_m.R' \setminus \{d_n\}} Sim(d_n, d_i))$$

Because

$$\mathsf{TRel}_{max}(G, d_n) \geq TRel(q_m, d_n)$$

and

$$\mathsf{Sim}_{min}(G, d_n) \leq \sum_{d_i \in q_m.R' \setminus \{d_n\}} Sim(d_n, d_i),$$

we complete the proof.

Proof of Lemma 5.

According to Equation 19, we have:

$$\widetilde{\mathsf{Sim}}_{min}(b, d_n) = e_s \times |\mathfrak{S}_w(b)| + minSim(U_w(b), d_n) \times (k - |\mathfrak{S}_w(b)|)$$

and

$$\widetilde{\mathsf{Sim}}'_{min}(b, d_n) = e_s \times |\mathfrak{S}'_w(b)| + minSim(U_w(b), d_n) \times (k - |\mathfrak{S}'_w(b)|).$$

Because $minSim(U_w(b), d_n) \leq e_s$ and $|\mathfrak{S}_w(b)| \geq |\mathfrak{S}'_w(b)|$, we complete the proof.

Proof of Lemma 6.

We have:

$$\sum_{d_j \in S} Sim(d_j, d_n) = \frac{1}{\parallel d_n.v_d \parallel} \sum_{d_j \in S \setminus \{d_e\}} \frac{\sum_{w_i \in d_j.v_d \wedge w_i \in d_n.v_d} d_j.v_d.w_i \times d_n.v_d.w_i}{\parallel d_j.v_d \parallel} = \frac{d_n.v_d.w_i}{\parallel d_n.v_d \parallel} \sum_{d_j \in S \setminus \{d_e\} \wedge w_i \in d_j.v_d} \frac{d_j.v_d.w_i}{\parallel d_j.v_d \parallel} = \frac{d_n.v_d.w_i}{\parallel d_n.v_d \parallel} \mathsf{AW}(w_i, S).$$

Proof of Lemma 4.

Since the query ids in a postings list are sorted in ascending order, if $p_w > b.maxID$, then based on the DAAT scheme it suggests that: (1) d_n cannot be a result of any query in b; or (2) queries in b that can have d_n as their results have already been found. Hence, unevaluated queries in b cannot contain any term that does not belong to the set $\{w_j|w_j \in d_n.\psi \wedge p_w \leq b.maxID\}$. So according to Equation 3 the text relevance between d_n and any query in b cannot exceed $max\{PS(d_n.v_d, w_i)|w_i \in d_n.\psi \wedge p_w \leq b.maxID\}$. So we complete the proof.

Proof of Theorem 1.

This problem can be reduced from the NP-hard *maximum independent set* problem. Given a graph $G(V, E)$, maximum independent set problem finds a set of vertices $V_m \subseteq V$ with maximum cardinality such that for every two vertices in V there is no edge connecting the two. Let E_v be a set of edges connecting to vertex v. We reduce maximum independent set problem to our problem by mapping each edge in E to a document in $U_w(b)$, and mapping E_{v_i} for each $v_i \in V$ to a set of documents S such that $\bigcup_{d_i \in S} Q_s(b, d_i) = b$.

Proof of Theorem 2.

Given a universe $U_w(b)$ and $\{Q_s(b, d_i)|d_i \in U_w(b)\}$, let $\mathcal{U}_b$ be the set of all MCSs that can be generated from $U_w(b)$, $\mathfrak{S}_w^{opt}(b) \subseteq \mathcal{U}_b$ be the set of disjoint MCSs with optimal cardinality, and $\mathfrak{S}_w^{G}(b) \subseteq \mathcal{U}_b$ be the set of disjoint MCSs generated by GreedyMcsGen. Let s be any fixed integer, and $\mathfrak{S}_w^{T}(b) \subseteq \mathcal{U}_b$ be any set of disjoint MCSs satisfying Condition (i).

(i) *For each $p \leq s$, the union of any $p+1$ disjoint sets among $\mathcal{U}_b$ intersects at least $p+1$ sets among $\mathfrak{S}_w^{T}(b)$.* According to the theorem proved by [20], we have:

$$|\mathfrak{S}_w^{opt}(b)|/|\mathfrak{S}_w^{T}(b)| \leq s_{max}/2 + \epsilon.$$

We next prove $\mathfrak{S}_w^{G}(b)$ satisfies Condition (i). Assume that $\mathfrak{S}_w^{G}(b)$ does not satisfy Condition (i). Then there must be an MCS $S \in \mathfrak{S}_w^{G}(b)$ and another MCS $S' \in \mathcal{U}_b$ such that $S' \subsetneqq S$. Consequently, based on Definition 5 S cannot be an MCS, which leads contradiction. As a result, we complete the proof.