Multi-Query Diversification in Microblogging Posts

Shiwen Cheng, Anastasios Arvanitis, Marek Chrobak, Vagelis Hristidis
Department of Computer Science and Engineering, University of California, Riverside, CA, USA
{schen064, tasos, marek, vagelis}@cs.ucr.edu

ABSTRACT

Effectively exploring data generated by microblogging services is challenging due to its high volume and production rate. To address this issue, we propose a solution that helps users effectively consume information from a microblogging stream, by filtering out redundant data. We formalize our approach as a novel optimization problem termed *Multi-Query Diversification Problem (MQDP)*. In MQDP, the input consists of a list of microblogging posts and a set of user queries (e.g. news topics), where each query matches a subset of posts. The objective is to compute the smallest subset of posts that cover all other posts with respect to a "diversity dimension" that may represent time or, say, sentiment. Roughly, the solution (cover) has the property that each covered post has nearby posts in the cover that are collectively related to all queries relevant to this covered post.

This is distinct from previous single-query diversity problems, as we may have two nearby posts that are related to intersecting but not nested sets of queries, in which case none covers the other. Another key difference is that we do not define diversity in terms of post similarity, since posts are too short for this approach to be meaningful; instead, we focus on finding representative posts for ordered diversity dimensions like time and sentiment, which are critical in microblogging. For example, for time as the diversity dimension, the selected posts will show how certain news events unfolded over time.

We prove that MQDP is NP-hard and we propose an exact dynamic programming algorithm that is feasible for small problem instances. We also propose two approximate algorithms with provable approximation bounds, and show how they can be adapted for a streaming setting. Through comprehensive experiments on real data, we show that our algorithms efficiently and effectively generate diverse and representative posts.

1. INTRODUCTION

User are overloaded by the high rate of produced microblogging posts, which often carry no new information with respect to other similar posts. Our work aims at developing a method for efficiently extracting diversified and representative posts from microblogging data. Here are some examples of applications that motivate our approach: *(i)* A user would like to subscribe to several queries (or topics or hashtags) in order to receive real-time posts relevant to her interests. For example a journalist that is interested in politics might follow a set of topics such as 'White House', 'senate' or 'Barack Obama', which can be represented as hashtags in a microblogging service like Twitter. Or, an investor might subscribe to a monitoring service that provides real time information relevant to terms such as 'GOOG', 'MSFT', or 'NASDAQ'. *(ii)* Alternatively, a user may search a microblogging site by submitting a set of queries instead of individual queries; this has been shown to improve the quality of search on documents [4].

In all these scenarios many microblogging posts will be relevant to the query topics, but the complete data set is likely to include multiple redundant posts with respect to dimensions such as time or sentiment. Sifting through such data would be overwhelming.

There has been work on building efficient indexes and search techniques for real-time search on microblogging data, such as EarlyBird [5], TI [6], and LSII [25]. However, these works do not address the information overload problem. There has been also extensive work on query results diversification [2, 3, 19, 10], where the key idea is to select a small set of posts that are sufficiently dissimilar, according to an appropriate similarity metrics.

For a number of reasons these diversity models are not quite adequate for multi-query searching or filtering applications as those described above: *(i)* Microblogging posts are too short for text distance functions to be effective – instead, we eliminate near-duplicate posts using existing duplicate detection methods like SimHash [17]. *(ii)* The query set is effectively guiding the content-based diversity, that is, the user expects to see some results for each of the query. *(iii)* Users may want to explore the data according to different diversity dimensions. Two such dimensions, especially relevant in microblogging, are time and sentiment, but other dimensions may also be useful in summarization of microblogging data.

In summary, our problem setting is fundamentally different from previous works on query results diversification in two ways: *(i)* In contrast to previous works that focus on results diversification for a single query, we study diversification with respect to multiple queries. In our setting, the user expresses her information need through a *set* of queries, for instance, by subscribing to a set of topics, like "Obama" or "economy." Since each post could be relevant to several queries, a post can be covered by a post in the results with respect to one query but not with respect to another. This motivates a *multiple-query definition of diversity coverage* where a post is covered only if it is covered with respect to *all* user-specified queries. *(ii)* Our diversity model does not use any inter-post similarity metric; instead, in our approach each post is assigned a value (e.g., timestamp) on the selected *diversity dimension*, and our method produces a subset of posts that covers the

10.5441/002/edbt.2014.13

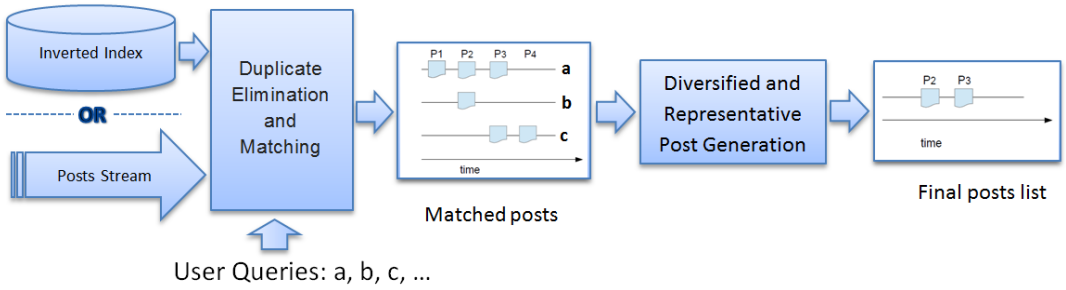


Figure 1: System Architecture. In this paper, we focus on the Diversified and Representative Post Generation part.

whole dimension range. As explained earlier, this model is more appropriate for the applications we are targeting.

To model this novel definition of diversity, we introduce an optimization problem called the *Multi-Query Diversification Problem (MQDP)*, defined as follows: Given a set of user queries, we aim to identify the minimum representative subset of microblogging posts such that *(i)* posts are diverse to each other (e.g., avoid posts matching the same query with similar sentiment, or publication time), and *(ii)* all posts that are relevant to at least one query are "covered" by a selected post. We define a *diversity threshold* λ on the diversity dimension, such that two posts at distance at most λ apart may cover each other (assuming they are associated to the same queries). E.g., the threshold can be 1 hour if the time dimension is selected.

We study two variations of MQDP. In the offline (static) version of MQDP, for a given dataset of microblogging posts, we seek to identify the minimum number of representative posts that cover every post in this dataset that is related to a set of queries (e.g. represented as hashtags). We show that MQDP is NP-hard, and we propose approximate algorithms to produce the representative results efficiently. In the streaming version of MQDP, we consider the scenario where posts arrive in a streaming fashion. The objective is to extract a small subset of posts that covers all the posts in the data stream, with the selected posts produced with a bounded delay.

A key challenge is that coverage is defined both on the diversity dimension and on the queries matched, that is, two posts relevant to different queries cannot cover each other, even if they have the same value on the diversity dimension. For example, a post that only matches query 'Obama' does not cover a post that only matches query 'economy', even if they have the same timestamp (assuming that time has been selected as the diversity dimension). Further, in the streaming variant, a key challenge is how to minimize the length of the returned diversified sub-stream, while at the same time incur a small delay in deciding if a new post should be returned or not. The naive approach would be to wait a long time after a post is published to be able to make a decision given its subsequent posts, but this would be unacceptable, as users expect very short delays when viewing microblogging data. We study these tradeoffs both theoretically and experimentally. Finally, we propose a principled approach to achieve *proportional diversity*, where we want to display to the user more posts from the more popular topics (queries), while at the same time maintaining diversity.

The system architecture is depicted in Figure 1, where time is selected as the diversity dimension. There are two options of providing input to the system. The first option, which corresponds to the Multi-Query Diversification problem, is by issuing a search query against an inverted index of microblogging posts. In the second option, corresponding to the Streaming Multi-Query Diversification problem, the matching module works directly on a stream of posts (e.g. Twitter stream).

Our contributions in this paper can be summarized as follows:

- We introduce and formalize the Multi-Query Diversification problem and its streaming variant (Section 2).
- We show that the Multi-Query Diversification problem is NP-hard (Section 3).
- We propose exact and approximation algorithms with provable approximation bounds for the Multi-Query Diversification problem and its streaming variant. We also study the tradeoff between the result size and the acceptable delay in returning a post for the streaming variant (Sections 4 and 5).
- We show a principled approach to achieve proportional diversity, where the popularity of topics (queries) is reflected in the result. For that, we show how a dynamic post-specific diversity threshold can be defined (Section 6).
- We conduct thorough experiments on real Twitter data and show that our proposed approximation algorithms can compute the solution efficiently and effectively (Section 7).

Section 8 presents related work, and we conclude and present future directions in Section 9.

2. PROBLEM FORMULATION

Definitions. Let L be a finite set of *labels* that can represent queries (such as hashtags or news articles) and $LP(a)$ be the list of microblogging posts that are relevant to a label $a \in \mathsf{L}$. Let P be the set of all posts.

We consider a diversity dimension F that defines a total order on the posts. We represent each post $P_i \in \mathsf{P}$ as a pair $(F(P_i), \mathit{label}(P_i))$ where $F(P_i)$ is the value of the post P_i in dimension F (for example, $F(P_i)$ can be the timestamp, or the sentiment polarity of post P_i) and $\mathit{label}(P_i) \subseteq \mathsf{L}$ is the set of labels that P_i is relevant to.

For ease of presentation and without harming the generality, in the remainder of this work we will assume that F represents the publication time of a post, i.e. $F(P_i) = \mathit{time}(P_i)$. Hence, we represent each post as a $P_i = (t_i, \mathit{label}(P_i))$ where $t_i = \mathit{time}(P_i)$ is the timestamp. If $t_i \leq t_j$, i.e., P_i is older than P_j we represent it as $P_i \prec_{time} P_j$. If both P_i and P_j are relevant to a label a and $|t_i - t_j| \leq \lambda$, then we will write that P_i *λ-covers* $a \in P_j$.

Example 1. *Consider the posts illustrated in Figure 2. If we define the threshold $\lambda = \Delta t$ then P_2 λ-covers $a \in P_1$ and $a \in P_3$, P_1 λ-covers $a \in P_2$, P_3 λ-covers $a \in P_2$, P_3 λ-covers $c \in P_4$, and P_4 λ-covers $c \in P_3$.*

All the above coverage examples are with respect to a single label. For our problem definition, we further define the *λ-cover* for a post and a set of posts as follows.

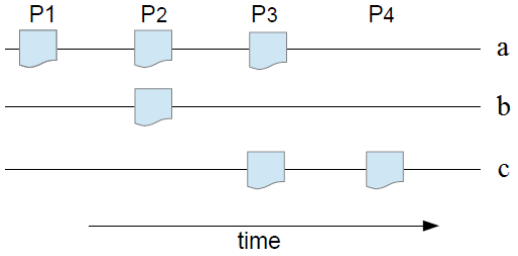


Figure 2: Example for coverage relations between posts. The time distances between each consecutive pair of posts are all Δt.

Definition 1. *(Post λ-cover) A post P_i is λ-covered by a set of posts Z if $\forall a \in label(P_i)$: $\exists P_j \in Z$ such that P_j λ-covers $a \in P_i$.*

Definition 2. *(Set λ-cover) Let P be a finite set of posts. $Z \subseteq \mathsf{P}$ is a λ-cover of P if $\forall P_i \in \mathsf{P}$: P_i is λ-covered by Z.*

Based on the above definitions, we formalize our Multi-Query Diversification Problem as follows:

Problem 1 [Multi-Query Diversification Problem (MQDP)] Given an instance $\langle \mathsf{P}, \lambda \rangle$: a collection of posts P and a distance threshold λ, compute the minimum cardinality subset of posts $Z \subseteq \mathsf{P}$ that λ-covers P.

Example 2. *Consider the posts depicted in Figure 2. Let $\mathsf{P} = \{P_1, P_2, P_3, P_4\}$. Again, assume $\lambda = \Delta t$. If we select P_2 and P_4 then $a \in P_1$ is λ-covered by P_2, $a \in P_3$ is λ-covered by P_2, and $c \in P_3$ is λ-covered by P_4. All posts have been λ-covered by P_2 or P_4, hence the set $\{P_2, P_4\}$ λ-covers P.*

In practice, posts might be arriving constantly. Hence, we also need to progressively report representative posts within a small time window from their publication timestamp.

Problem 2 [Streaming Multi-Query Diversification Problem (StreamMQDP)] Given a set of labels (e.g. queries) L, a set P of incoming posts where each post P_j arrives at timestamp t_j, and a distance threshold λ, progressively report a small cardinality $Z \subseteq \mathsf{P}$ that λ-covers P, with the constraint that each post $P_i \in Z$ needs to be reported within time τ from *time*(P_i).

In general we might want to diversify incoming posts based on a function other than their publication time, e.g. based on their sentiment polarity or distance from a user's location. The above definition of the problem can accommodate this scenario. However in this setting post coverage will be computed based on the respective distance function defined on sentiment polarity or geolocation.

3. NP-HARDNESS OF MQDP

If all posts in an instance of MQDP are issued at exactly the same time, then this instance is in essence an instance of the set cover problem, where the sets are the queries (represented as sets of labels). This immediately implies NP-hardness of MQDP; in fact, it also implies that MQDP cannot be approximated within ratio better than $\ln |L|$ [12] (under appropriate complexity-theoretic assumptions). However, the instances needed for this hardness proof require queries with arbitrary number of labels and such instances would not appear in realistic data sets. In this section we show that MQDP remains NP-hard even for instances with few labels per post.

Lemma 1. *MQDP is NP-hard, even for instances with at most two labels per post.*

Proof. We show that CNF (the satisfiability problem for conjunctive normal form formulas) reduces to MQDP in polynomial time. Let $\alpha = C_1 \wedge ... \wedge C_m$ be a CNF formula with variables $x_1, ..., x_n$, where $C_1, ..., C_m$ are clauses. We transform α into an instance $\langle \mathsf{P}, \lambda \rangle$ of MQDP such that α is satisfiable if and only if P has a λ-cover of cardinality at most $n(2m+3)$.

We now describe this construction. We will take $\lambda = 1$, and the set of labels will be $\mathsf{L} = \{w_i, u_i, \bar{u}_i\}_{i=1,...,n} \cup \{c_j\}_{j=1,...,m}$. We will have posts issued at all integral times $1, ..., 2m+3$. Specifically, for each $i = 1, ..., n$, P will contain the following posts:

(i) $(1, \{u_i, w_i\})$, $(1, \{\bar{u}_i, w_i\})$,

(ii) $(2m+3, \{u_i, w_i\})$, $(2m+3, \{\bar{u}_i, w_i\})$, and

(iii) $(2j, \{u_i\})$, $(2j, \{\bar{u}_i\})$, for all $j = 1, ..., m+1$.

Also, for each $i = 1, ..., n$ and $j = 1, ..., m$, we include posts $(2j+1, U_{ij})$ and $(2j+1, \bar{U}_{ij})$, whose label sets U_{ij} and $\bar{U}_{ij}$ depend on whether clause C_j contains variable x_i or its negation:

(iv) If $x_i \in C_j$ then $U_{ij} = \{u_i, c_j\}$, else $U_{ij} = \{u_i\}$.

(v) If $\bar{x}_i \in C_j$ then $\bar{U}_{ij} = \{\bar{u}_i, c_j\}$, else $\bar{U}_{ij} = \{\bar{u}_i\}$.

There are no other posts in P. (See Figure 3 for an example.)

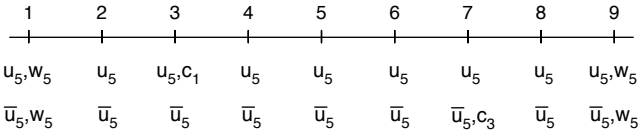


Figure 3: An illustration of the construction where $m = 3$, showing the posts for $i = 5$. Only the label sets are shown, to avoid clutter. The example assumes that $x_5 \in C_1$, $\bar{x}_5 \in C_3$, and that these are the only occurrences of x_5 in α.

This reduction clearly runs in polynomial time. So to complete the proof it is sufficient to show the following claim: α is satisfiable if and only if P has a λ-cover of cardinality at most $n(2m+3)$. To prove this claim, we consider both implications separately.

($\Rightarrow$) Suppose that α is satisfiable by some truth assignment $f()$. The corresponding 1-cover Z for P is constructed as follows. If $f(x_i) = 1$, we include the following posts in Z; $(1, \{u_i, w_i\})$, $(2m+3, \{u_i, w_i\})$, $(2j, \{\bar{u}_i\})$ for all $j = 1, ..., m+1$, and $(2j+1, U_{ij})$ for all $j = 1, ..., m$. If $f(x_i) = 0$, we include the following posts in Z: $(1, \{\bar{u}_i, w_i\})$, $(2m+3, \{\bar{u}_i, w_i\})$, $(2j, \{u_i\})$ for all $j = 1, ..., m+1$, and $(2j+1, \bar{U}_{ij})$ for all $j = 1, ..., m$. For each i we thus include $2 + m + 1 + m = 2m + 3$ posts, for the total of $n(2m+3)$. All labels w_i, u_i, and $\bar{u}_i$ are easily seen to be covered. We claim that all labels c_j are covered as well. Consider any label c_j. Since α is satisfied by $f()$, at least one literal in the corresponding clause C_j is true. Suppose that $x_i \in C_j$ satisfies C_j, that is $f(x_i) = 1$. (If C_j is satisfied by $\bar{x}_i$, the argument is similar.) Then, by the definition of Z, post $(2j+1, U_i)$ is in Z and $U_i = \{u_i, c_j\}$. All occurrences of c_j are at time $2j+1$, so c_j is covered by Z, as claimed.

($\Leftarrow$) Now suppose that P has a 1-cover Z of cardinality $n(2m+3)$. Consider a subset Z_i of Z consisting of all posts that contain

labels w_i, u_i or $\bar{u}_i$, that is Z_i contains all posts from Z of the following form:

- $(1, \{u_i, w_i\})$, $(1, \{\bar{u}_i, w_i\})$,
- $(2m+3, \{u_i, w_i\})$, $(2m+3, \{\bar{u}_i, w_i\})$,
- $(2j, \{u_i\})$, $(2j, \{\bar{u}_i\})$, for $j = 1, ..., m+1$, and
- $(2j+1, U_{ij})$, $(2j+1, \bar{U}_{ij})$, for $j = 1, ..., m$.

We claim that $|Z_i| \geq 2m+3$. There are $2m+3$ posts with u_i, at times $1, 2, ..., 2m+3$, so to cover them all we need at least $m+1$ posts, and the only way to cover them with $m+1$ posts is by choosing posts $(2j, \{u_i\})$, for $j = 1, ..., m+1$. Similarly, we need at least $m+1$ posts to cover all $\bar{u}_i$'s, and the only way to do that with $m+1$ posts would be to choose posts $(2j, \{\bar{u}_i\})$, for $j = 1, ..., m+1$. Note that these two sets of posts are disjoint and together they have $2m+2$ posts, with all w_i's are still uncovered. Thus Z_i must indeed contain at least $2m+3$ posts.

Since $|Z_i| \geq 2m+3$ for all $i = 1, ..., n$ and our budget for posts is $n(2m+3)$, we must have that in fact $|Z_i| = 2m+3$ for all i. Consider any i. To cover all w_i's, Z_i must include at least one of $(1, \{u_i, w_i\})$, $(1, \{\bar{u}_i, w_i\})$ and at least one of $(2m+3, \{u_i, w_i\})$, $(2m+3, \{\bar{u}_i, w_i\})$. As covering all u_i's requires $m+1$ posts and covering all $\bar{u}_i$'s required $m+1$ different posts, So Z_i must have $m+1$ posts covering one of u_i, $\bar{u}_i$ and $m+2$ posts covering the other, with these other posts covering also all w_i's. We thus obtain that there are only two choices for Z_i:
$\{(1, \{u_i, w_i\}), (2m+3, \{u_i, w_i\})\}$
$\cup \{(2j+1, U_{ij})\}_{j=1}^{m} \cup \{(2j, \bar{u}_i)\}_{j=1}^{m+1}$ or
$\{(1, \{\bar{u}_i, w_i\}), (2m+3, \{\bar{u}_i, w_i\})\}$
$\cup \{(2j+1, \bar{U}_{ij})\}_{j=1}^{m} \cup \{(2j, u_i)\}_{j=1}^{m+1}$.

With the above in mind, we show that α must be satisfiable. If Z_i is of the first type, we set $f(x_i) = 1$, and if Z_i is of the second type then we set $f(x_i) = 0$. Let C_j be any clause. We need to show that C_j is satisfied. In Z the corresponding label c_j is covered, which means that Z contains some post $(2j+1, U_{ij})$ or $(2j+1, \bar{U}_{ij})$ which contains c_j. By symmetry, we can assume that $c_j \in U_{ij}$. By the form of Z_i, as described above, that means that $x_i \in C_j$. The correspondence between $f()$ and Z implies also that $f(x_i) = 1$. Thus x_i satisfies clause C_j. □

4. ALGORITHMS FOR MQDP

A naive, exhaustive search algorithm to optimally solve MQDP would run in time exponential in $|\mathsf{P}|$, the number of posts. We first show that the computational complexity of this algorithm can be significantly reduced using dynamic programming, to running time that is exponential only in $|\mathsf{L}|$, the number of labels, which in practice is a small integer. To reduce the running time even further, we present two polynomial-time algorithms that produce approximate solutions. The first one is inspired by solutions to the set cover problem, which has an approximation bound of $ln(|\mathsf{P}||\mathsf{L}|)$. The second one is a novel algorithm based on a traversal of the ordered (by the diversity dimension) list of input posts, which has a tighter approximation bound of s, where s is the maximum number of labels (queries) that a post may be associated with.

4.1 Algorithm OPT

We now propose an exact algorithm based on dynamic programming, which we refer to as *OPT*. We start with several definitions.

Definitions. Number the posts $P_1, P_2, ..., P_n$ ordered by their timestamps. Letting $t_j = \mathit{time}(P_j)$ for all j, we then have $t_1 < t_2 < ... < t_n$. (We assume all posts timestamps are different, for simplicity.) To simplify the description of the algorithm, we further assume that we have an additional initial post P_0 that contains all the labels, i.e., $\mathit{label}(P_0) = \mathsf{L}$. Any instance can be modified to have this property, by adding a new post with all labels and $\epsilon > \lambda$ time unit before the first post. This new post will have to belong to any solution, and it increases the optimum solution by exactly one post element.

We need a few other definitions. For any $j = 1, ..., n$, let $f(j) = \max\{j' \geq j : t_{j'} \leq t_j + \lambda\}$. Define a (λ, j)-*cover* to be a set of posts $Z \subseteq \{P_1, ..., P_{f(j)}\}$ that covers all posts $P_1, ..., P_j$. Note that we do not need to include any posts after time $t_{f(j)}$, because they cannot cover any posts $P_1, ..., P_j$.

The *end-pattern* of a (λ, j)-cover Z is defined as a function $\xi : \mathsf{L} \to \{1, 2, ..., f(j)\}$ that to each $a \in \mathsf{L}$ assigns the index $\xi(a)$ of the latest post $P_{\xi(a)}$ in Z that contains a. More precisely, we have $a \in \mathit{label}(P_{\xi(a)})$, $P_{\xi(a)} \in Z$ and $a \notin \mathit{label}(P_i)$ for each $P_i \in Z$ such that $i > \xi(a)$.

If ξ is the end-pattern of some (λ, j)-cover then we will refer to ξ as a *j-end-pattern*. It is easy to see that **a function $\xi : \mathsf{L} \to \{1, 2, ..., f(j)\}$ is a j-end-pattern if and only if it satisfies the following conditions for each label $a \in \mathsf{L}$**:

(i) For any $b \in \mathsf{L}$, if $\xi(b) > \xi(a)$ then $a \notin \mathit{label}(P_{\xi(b)})$.

(ii) if $t_{\xi(a)} + \lambda < t_i \leq t_j$ then $a \notin \mathit{label}(P_i)$.

We will denote by Ξ_j the set of all j-end-patterns. An example illustrating the definition of end-patterns is shown in Figure 4.

The algorithm proceeds from left to right, one post at a time. When processing each P_j, the algorithm will keep track of a set of partial solutions that cover the first j posts $P_1, ..., P_j$. This set of partial solutions is chosen so that at least one of them can be extended to a global optimal solution. On the other hand, we need to make this set small to obtain good running time.

Specifically, for each j and each $\xi \in \Xi_j$ we will compute the cardinality $h_{j,\xi}$ of the optimal (λ, j)-cover with end-pattern equal ξ. Initially, by our assumption about post P_0, the only 0-end-pattern in Ξ_0 is ξ defined by $\xi(a) = 0$ for all $a \in \mathsf{L}$. For this ξ, we set $h_{0,\xi} = 1$.

Next, suppose that $j \geq 1$ and that we have all values $h_{j-1,\eta}$, for $\eta \in \Xi_{j-1}$, already computed. Then, for each $\xi \in \Xi_j$, $h_{j,\xi}$ is computed according to the following formula:

$$h_{j,\xi} = \min_{\substack{\eta \in \Xi_{j-1} \\ \eta \preceq \xi}} \{h_{j-1,\eta} + \Delta(\eta, \xi)\}. \quad (1)$$

We now explain the notations used in this formula:

- $\eta \preceq \xi$ means that η is consistent with ξ, that is, for any $a \in \mathsf{L}$, if $\xi(a) \leq f(j-1)$ then $\xi(a) = \eta(a)$.
- $\Delta(\eta, \xi) = |\{P_{\xi(a)} : \xi(a) > f(j-1)\}|$ is the number of posts in ξ that are not in η.

When the above iteration completes, the algorithm outputs $\min_{\xi \in \Xi_n} h_{n,\xi}$ as the optimum value.

Implementation. We present the algorithm in pseudocode as Algorithm 1, where we return the minimum cardinality instead of the final posts list for simplicity. Algorithm 1 can be easily modified to return the final posts if we maintain all j-end-patterns for each j such that we get the final list of posts by backtracking.

Algorithm 1 starts from the first post P_1. At step j (working on post P_j) we already have Ξ_{j-1}, and we first generate all candidate j-end-patterns, denoted $\hat{\Xi}_j$ (lines 4 - 9). For this, we only need

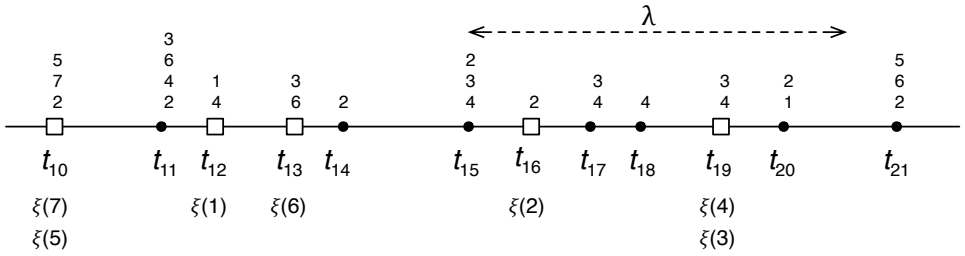


Figure 4: An example of an end-pattern. In this figure the label set is $\mathsf{L} = \{1, 2, 3, 4, 5, 6, 7\}$. A $(\lambda, 15)$-cover $Z = \{..., P_{10}, P_{12}, P_{13}, P_{16}, P_{19}\}$ is marked with squares. The 15-end-pattern of Z is $\xi(1, 2, 3, 4, 5, 6, 7) = (12, 16, 19, 19, 10, 13, 10)$.

Algorithm 1 Algorithm OPT

Input: A list of microblogging posts P sorted by timestamp, each post $P_j \in \mathsf{P}$ with a set of labels *label*(P_j), a threshold λ.
Output: The minimum cardinality of a λ-cover P.

```
1: Ξ_0 = {(0, ..., 0)}
2: H_0((0, ..., 0)) = 1
3: for j = 1 → |P| do
4:     ppl ← {} // posts per label
5:     for a ∈ |L| do
6:         ppl[a] ← posts in LP(a) in [t_j − λ, t_j + λ]
7:         if a ∉ label(P_j) then
8:             ppl[a].add(0)
9:     Generate j-end-patterns: for each label a get one item from ppl[a],
       then form an end-pattern with these |L| items. If it is valid (see
       conditions of j-end-pattern), add this end-pattern to Ξ̂_j.
10:    for ξ̂ ∈ Ξ̂_j do
11:        for η ∈ Ξ_{j−1} do
12:            ξ ← ξ̂
13:            if ∃a ∈ L : η(a) ≠ ξ(a) ∧ 0 < ξ(a) ≤ f(j − 1) then
14:                next // in this case, it is impossible that η ⪯ ξ
15:            Δ ← ∅
16:            for a ∈ L do
17:                if η(a) ≠ ξ(a) ∧ ξ(a) ≠ 0 ∧ ξ(a) ∉ Δ then
18:                    Δ.add(ξ(a))
19:                if ξ(a) ≠ η(a) ∧ ξ(a) == 0 then
20:                    ξ(a) ← η(a) // handle the 0 items in ξ
21:            if ξ is not valid then
22:                next // see conditions of j-end-pattern
23:            card(ξ) = H_{j−1}(η) + |Δ|
24:            if ξ ∈ Ξ_j then
25:                H_j(ξ) ← min{card(ξ), H_j(ξ)}
26:            else
27:                Ξ_j.add(ξ)
28:                H_j(ξ) ← card(ξ)
29: return min_{ξ∈Ξ_{|P|}} H_{|P|}(ξ) − 1
```

to check the posts in the range $[t_j - \lambda, t_j + \lambda]$ as posts before this range cannot cover P_j nor later posts, so they do not affect the subsequent computation. For each candidate end-pattern $\hat{\xi}$ in $\hat{\Xi}_j$, $\forall a \in$ *label*(P_j) : $P_{\hat{\xi}(a)}\lambda$-covers $a \in$ *label*(P_j), and for a label $a \notin$ *label*(P_j), $\hat{\xi}(a)$ could be 0 (which means that we do not have to select any post for the labels not in *label*(P_j) to λ-covers P_j), or any i that $a \in$ *label*(P_i) with $|t_i - t_j| \leq \lambda$ (lines 7 - 9). This case ($\hat{\xi}(a) = 0$) will be fixed during the computation of the cardinality of a j-end-pattern based on Ξ_{j-1}, where we only considers $\eta \in \Xi_{j-1}$ that $\eta \preceq \xi$ (lines 13 - 14 ensure this) and we update the $\xi(a)$ to $\eta(a)$ where $\xi(a) = 0$ (lines 19 - 20). We put the updated end-pattern into Ξ_j, and save or update its cardinality $h_{j,\xi}$ in H_j (lines 24 - 28) where $H_j(\xi)$ is the optimal cardinality of each end-pattern ξ in Ξ_j.

Correctness. To prove correctness, we first show feasibility, namely that for each j and $\xi \in \Xi_j$ there is a (λ, j)-cover with end-pattern equal to ξ and cardinality $h_{j,\xi}$. This can be shown by simple induction on j. Suppose that the claim holds for $j - 1$. Fix any $\xi \in \Xi_j$. For this ξ, let $\eta \in \Xi_{j-1}$ be the end-pattern that realizes the minimum in (1). By the inductive assumption, there is a $(\lambda, j-1)$-cover Y with end-pattern η and cardinality $h_{j-1,\eta}$. By adding to Y the posts $P_{\xi(a)}$ for $a \in \mathsf{L}$ such that $\xi(a) > f(j-1)$, we obtain a (λ, j)-cover Z with end-pattern ξ and $h_{j-1,\eta} + \Delta(\eta, \xi) = h_{j,\xi}$ posts. Note that Z is indeed a correct (λ, j)-cover, since all posts $P_1, ..., P_{j-1}$ are covered by Y and P_j is covered by Z, by the definition of ξ.

Next, we argue that the final solution is indeed optimal. It is sufficient to prove that for each j and $\xi \in \Xi_j$ the value of $h_{j,\xi}$ is optimal. We again proceed by induction on j. Assume the claim holds for $j - 1$. Fix any $\xi \in \Xi_j$ and define Z^* to be an optimal (λ, j)-cover with end-pattern ξ. Let Y^* be obtained from Z^* by removing the posts (strictly) after $t_{f(j-1)}$ and let η be the end-pattern for Y^*. Then $\eta \preceq \xi$ and Y^* must be an optimal $(\lambda, j-1)$-cover with end-pattern η, since otherwise, if a smaller $(\lambda, j-1)$-cover with end-pattern η existed, we could add to it the posts of ξ after $t_{f(j-1)}$ and obtain a (λ, j)-cover with end-pattern ξ with smaller cardinality than Z^*. Thus Y^* has cardinality $h_{j-1,\eta}$, by the inductive assumption. Consequently, the cardinality of Z^* is $h_{j-1,\eta} + \Delta(\eta, \xi)$.

Time Complexity. The number of all end-patterns is $O(|\mathsf{P}|^{|\mathsf{L}|})$, so the loop on j and ξ will iterate $O(|\mathsf{P}|^{|\mathsf{L}|+1})$ times. Computing the minimum in (1) takes time $O(|\mathsf{P}|^{|\mathsf{L}|})$ as well, so the time complexity will be $O(|\mathsf{P}|^{2|\mathsf{L}|+1})$.

Space Complexity. The most expensive part in terms of required space is on saving all end-patterns at each position. In order to compute the minimum cardinality, we only need to maintain the patterns for the current position and the previous position, as shown in Algorithm 1. So the space complexity will be $O(|\mathsf{P}|^{|\mathsf{L}|})$. Finally we want to get the minimum set of posts, so we need to keep the end-patterns at every position to be used for backtracking. Thus, the space complexity will be $O(|\mathsf{P}|^{|\mathsf{L}|+1})$.

4.2 Algorithm GreedySC

With running time $O(|\mathsf{P}|^{2|\mathsf{L}|+1})$, Algorithm OPT may still be impractical for large data sets. Thus we propose approximation algorithms to solve this problem more efficiently.

The first approach is to transform an MQDP instance $\langle \mathsf{P}, \lambda \rangle$ to a set cover problem instance and then apply the greedy set-cover algorithm. Each element of thus constructed set cover problem instance is a pair of a post $P_i \in \mathsf{P}$ and a label a: $\langle P_i, a \rangle$ where $a \in$ *label*(P_i). Thus, the universe of the set cover instance is $U = \{\langle P_i, a \rangle\}_{i=1,...,|\mathsf{P}|, a \in label(P_i)}$. We have $|\mathsf{P}|$ sets in the set cover problem (one set for each post). The k-th set S_k contains the set of pairs that are λ-covered by picking P_k, i.e.

Algorithm 2 Algorithm GreedySC

Input: A list of microblogging posts P sorted by timestamp, a set of labels L, each label $a \in \mathsf{L}$ with a list of relevant posts $LP(a) \subseteq P$ sorted by timestamp, a threshold λ.
Output: A subset of posts $Z \subseteq \mathsf{P}$, such that Z λ-covers P.

$S = \{S_1, S_2, ..., S_{|\mathsf{P}|}\}$, initiate each set $S_i \in S$ as $\emptyset$
$Z = \emptyset$
for $a \in \mathsf{L}$ **do**
  for $j = 1 \rightarrow |LP(a)|$ **do**
    for $i = j \rightarrow |LP(a)|$ **do**
      if $|time(LP(a)[j]) - time(LP(a)[i])| > \lambda$ **then**
        break
      $x \leftarrow$ index of $LP(a)[i]$ in P
      $y \leftarrow$ index of $LP(a)[j]$ in P
      $S_x.add(\langle LP(a)[j], a\rangle)$
      $S_y.add(\langle LP(a)[i], a\rangle)$
while true **do**
  $i \leftarrow \arg\max_x(|S_x|)$
  if $|S_i| == 0$ **then**
    break
  $Z.add(P_i)$
  for $j = 1 \rightarrow |\mathsf{P}|$ **do**
    $S_j = S_j - S_i$
return Z

$S_k = \bigcup_{a \in label(P_k)} \{\langle P_i, a\rangle : a \in label(P_i), |t_k - t_i| \leq \lambda\}$.

Algorithm 2 depicts this approach. For ease of presentation we refer this algorithm as *GreedySC* in the remainder of this paper. At each iteration, GreedySC selects the set that contains the largest number of yet uncovered elements.

Approximation bound. This algorithm has an approximation ratio of $\ln k$, where k is the maximum set size [12]. In our case, $k \leq |\mathsf{P}||\mathsf{L}|$, so $|S^{GreedySC}| \leq (\ln|\mathsf{P}| + \ln|\mathsf{L}|)|S^{opt}|$. In practice, $|\mathsf{P}|$ is much larger than $|\mathsf{L}|$ and hence the error bound is essentially $\ln|\mathsf{P}|$.

4.3 Algorithm Scan

We now propose another algorithm with a provable approximation bound and better running time. The algorithm process the relevant posts of each label separately. It *scans* each sorted list $LP(a)$ to find the optimal solution S_a in terms of label a, and it outputs $S^{scan} = \bigcup_{a \in L} S_a$ as its final solution.

For each label a, the scan starts from the first post in $LP(a)$. During the scan, we keep track of the most recent uncovered post P_x. Thus, initially P_x is the first post. We scan forward until finding a post P_y such that $|t_x - t_y| > \lambda$. Then we pick the post P_z that is right before P_y and add it to S_a. Then the posts from P_x to P_z can be all marked as covered (in terms of label a). With the scan continues, we mark the posts within distance of λ to P_z as covered (in terms of label a) and we reset P_x to be the first post that has distance larger than λ to P_z. The algorithm continues with the above procedure until it reaches the end of the list, then if the last post is not λ-covered by a selected post, then we add it to S_a. The pseudocode is presented in Algorithm 3.

Correctness. For each post P_i in $LP(a)$ there must exist a post in S_a that λ-covers $a \in P_i$ with Algorithm Scan. Since S^{scan} is the union of S_a for all $a \in \mathsf{L}$, then S^{scan} λ-covers each $P_i \in \mathsf{P}$. Thus S^{scan} is a λ-cover of P.

Approximation bound. Assume that each post is relevant with at most $s \leq |\mathsf{L}|$ labels. Then the approximation bound of Algorithm Scan is s, i.e. $|S^{scan}| \leq s|S^{opt}|$.

Proof. It is routine to prove that S_a is an optimal λ-cover of $LP(a)$. Thus, we have $|LP(a) \cap S^{opt}| \geq |S_a|$. Hence $\sum_{a \in \mathsf{L}} |LP(a) \cap S^{opt}| \geq \sum_{a \in \mathsf{L}} |S_a|$. Since each post relates with at most $s \leq |\mathsf{L}|$ labels, thus $s|S^{opt}| \geq \sum_{a \in \mathsf{L}} |LP(a) \cap S^{opt}| \geq \sum_{a \in \mathsf{L}} |S_a|$. Further it holds that $\sum_{a \in \mathsf{L}} |S_a| \geq \bigcup_{a \in \mathsf{L}} |S_a| = |S^{scan}|$. It follows that $|S^{scan}| \leq s|S^{opt}|$. □

Algorithm 3 Algorithm Scan

Input: A set of labels L, each label $a \in \mathsf{L}$ with a list of relevant posts $LP(a)$ sorted by timestamp, a threshold λ.
Output: A subset of posts $Z \subseteq \mathsf{P}$, such that Z λ-covers P.

$Z = \emptyset$
for $a \in \mathsf{L}$ **do**
  if $|P(a)| == 0$ **then**
    next
  $j \leftarrow 1$
  $left \leftarrow LP(a)[j]$
  $picked \leftarrow null$
  while $j \leq |LP(a)|$ **do**
    if $|time(LP(a)[j]) - time(left)| \leq \lambda$ **then**
      $j \leftarrow j + 1$
    else
      $picked \leftarrow LP(a)[j-1]$
      $Z.add(picked)$
      while $j \leq |LP(a)|$ **do**
        if $|time(LP(a)[j]) - time(picked)| \leq \lambda$ **then**
          $j \leftarrow j + 1$
        else
          $left \leftarrow LP(a)[j]$
          break
  $last \leftarrow$ the last post in $LP(a)$
  if $picked == null \vee |time(last) - time(picked)| > \lambda$ **then**
    $Z.add(last)$
return Z

Time Complexity. Algorithm Scan examines each post in $LP(a)$ only once, thus the running time is $\sum_{a \in \mathsf{L}} |LP(a)|$, which is $O(s|\mathsf{P}|)$.

Optimizations of Algorithm Scan. Algorithm Scan processes each label separately, which results in some inefficiency, because the posts selected for one label may also cover posts from other labels. We consider an improvement to address this inefficiency. When selecting a post P_i for a label a, we remove all posts covered by P_i from all subsequent lists $LP(b)$. The effectiveness of this optimization depends on the ordering of the labels processed by Scan. We refer to this variant of Scan as *Scan+*.

5. ALGORITHMS FOR STREAMMQDP

For StreamMDQP, the posts/label matching module works directly on the stream of microblogging posts instead of a collection of posts that have been indexed. The algorithm selects a subset stream of the posts to λ-cover the whole stream. For a new relevant post, the algorithm waits a small time τ to make the decision whether output this new post or not. On one hand, to minimize the delay, we want to make a decision as soon as possible on whether a post should be outputted or not. On the other hand, a longer delay increases the probability of finding a smaller cover. Ideally, we should decide if a post should be output immediately, that is, with delay $\tau = 0$. However, as we show, this leads to an increased total number of output posts.

We show how Algorithm GreedySC and Algorithm Scan can be adapted for a streaming setting.

5.1 Streaming Scan

We show that when we make an instant decision ($\tau = 0$), we incur an error of up to $2s$, whereas if we have a delay of $\tau \geq \lambda$ then we have the original Algorithm Scan error bound of s.

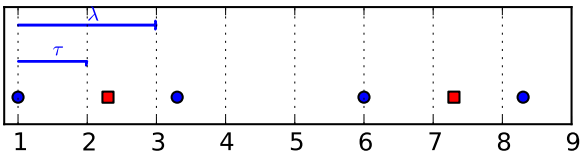


Figure 5: An example with approximation ratio 2, for $0 < \tau < \lambda$

Delayed ouput of a new post. As we process the stream in chronological order, we keep the list of posts that have been already outputted. New posts can be added to the output if they are not λ-covered by the previously selected ones. In order to decide whether a new post has to be included in the results we can apply Algorithm Scan, which is natural for streaming environments, since it processes the posts in order.

The algorithm, for each label $a \in \mathsf{L}$, keeps track of the following posts: the oldest and the latest uncovered relevant posts, denoted as $P^{ou}(a)$ and $P^{lu}(a)$ respectively, and the latest outputted relevant post $P^{lc}(a)$. And the algorithm waits until time $\min\{\textit{time}(P^{lu}(a)) + \tau, \textit{time}(P^{ou}(a)) + \lambda\}$ to output $P^{lu}(a)$, at the same time set $P^{lc}(a)$ to $P^{lu}(a)$ and both $P^{ou}(a)$ and $P^{lu}(a)$ to null. When a new post P_x arrives, for each label $a \in \textit{label}(P_x)$, if $P^{lc}(a)$ λ-covers $a \in P_x$, the algorithm doesn't update anything for label a. Or else the algorithm sets $P^{lu}(a)$, as well as $P^{ou}(a)$ if it is originally null, to P_x. We denote this algorithm as *StreamScan*.

Similar to the improvement of Algorithm Scan by Scan+, we can apply the same idea to StreamScan as *StreamScan+*.

Approximation bound. This algorithm outputs posts exactly as Algorithm Scan when $\tau \geq \lambda$, thus they have the same approximation error bound, i.e. s.

Instant output of a new post. Instead of waiting up to τ before a post to be outputted, we can directly decide whether to output a post or not right after it arrives. For this, we use a small cache that keeps track of the most recently selected posts for each label $a \in \mathsf{L}$. When a new relevant post P_x arrives, if P_x is not covered, we output P_x and update the cache appropriately.

Approximation bound. We can show that the above algorithm achieves a $2s$ approximation bound if each post is relevant to at most s labels when $0 \leq \tau < \lambda$.

In order to prove this, we will firstly consider the case when there is only one label. Assume that the results from Algorithm StreamingScan consists of posts $P_{i_1}, P_{i_2}, ..., P_{i_n}$ with timestamps $t_{i_1} < t_{i_2} < ... < t_{i_n}$. For $1 \leq j < n$, the distance between t_{i_j} and $t_{i_{j+1}}$ is larger than λ (otherwise they will cover each other and the algorithm will not pick both of them). Then $t_{i_{j+2}} - t_{i_j} > 2\lambda$ thus no post can λ-cover both P_{i_j} and $P_{i_{j+2}}$. Hence, an optimal solution must contain no less than $n/2$ posts. Thus the size of solution from this algorithm for label a, $S_a \leq 2S_a^{opt}$, where S_a^{opt} denotes the optimal solution size for the single label a.

We show an example in Figure 5 for $0 < \tau < \lambda$, where the optimal solution consists of the posts presented by red squares, and the algorithm will return the posts presented as blue dots. Thus the error factor is 2.

Recall the analysis of approximation bound of Algorithm Scan, the sum of optimal solution size from each single label is less than s times of global optimal solution size, i.e. $\sum_{a \in \mathsf{L}} |S_a^{opt}| \leq sS^{opt}$, where S^{opt} is the global optimal solution. Thus StreamingScan has a $2s$ approximation bound in the case of $0 < \tau < \lambda$.

5.2 Streaming Version of GreedySC

Delayed ouput of a new post. If we can tolerate a delay of $\tau > 0$, then the streaming Set-cover-based algorithm works as follows: Assume P' is the oldest post that has not been covered yet, and assume its timestamp is $\textit{time}(P')$. Then, we wait until time $\textit{time}(P') + \tau$; let Z be the set of posts with timestamp between $\textit{time}(P')$ and $\textit{time}(P') + \tau$. We execute the GreedySC algorithm in Z selecting posts to output, until posts in Z are all covered. We pass the ones that are already covered and set again $\textit{time}(P')$ to be the oldest uncovered post (the first post in subsequent stream that is not λ-covered by selected posts), and we repeat the same procedure. We denoted this algorithm as *StreamGreedySC*.

We can have a variation of StreamGreedySC: instead of executing GreedySC on Z until all posts are covered, we can stop GreedySC on Z once P' is covered and then update the oldest uncovered post P' (which is possibly in Z). We refer this variation as *StreamGreedySC+*.

Instant ouput of a new post. If we want to instantly make a decision on whether to output a post, i.e. $\tau = 0$, then the streaming version of the Set-cover-based algorithm is the same as the one for Algorithm Scan, which has worst case error bound of $2s$, as explained above.

6. PROPORTIONAL DIVERSITY THROUGH VARIABLE λ

In the previous sections, we studied the Multi-Query Diversification problem assuming that the coverage parameter λ is applied uniformly for the whole range. However, this does not address the problem of *representativeness* of the results. For example, if many posts are posted in the morning but few in the afternoon, it may be desirable to return more morning posts in the diversified result. Similarly, if we focus on the sentiment diversity, the selected posts must reflect the distribution of the public's sentiment. For example, news about a decrease in the national unemployment rate would receive more positive posts, and hence we want to display more positive posts.

We propose to consider a different λ for each post, such that λ is larger in sparse areas and smaller in dense areas. The intuition is that if there are many say negative posts, then a negative post should cover another negative post only if they are very close to each other in terms of sentiment score. This will lead to displaying more negative posts, to better represent the complete set of posts.

More specifically, we define a λ value for each pair of post and label that this post matches, i.e., for each post $P_i \in \mathsf{P}$ and label $a \in \textit{label}(P_i)$ we define $\lambda_a(P_i)$, proportional to the density of posts around P_i that match label a.

However, we don't want the variation of λ to be too drastic, because then rare perspectives would not get represented. For example, if there are 20 negative posts and 2 positive, and we only show 3 to the user, it would make sense to also show one positive one. For this reason, it makes sense to choose λ to be a non-linear function of post density. We propose here a smooth diversity formula, inspired by the work in [26] on single-query searches, with $\lambda_a(P_i)$ defined by:

$$\lambda_a(P_i) = \lambda_0 e^{1 - \frac{density_a(t_i - \lambda_0, t_i + \lambda_0)}{density_0}}, \qquad (2)$$

where λ_0 is a constant threshold set by a domain expert, $density_a(t_i - \lambda_0, t_i + \lambda_0)$ is the density of posts that match label a in the time range $[t_i - \lambda_0, t_i + \lambda_0]$ (i.e., the number of posts per minute matching label a in this time range) and $density_0$ is the average density of posts in a $2\lambda_0$ time interval across all labels (i.e., the average number of posts per minute relevant to any label $a \in \mathsf{L}$).

By applying Equation 2, we achieve proportional diversity in terms of both *(i)* labels, i.e., the output will contain more posts

from the labels with larger number of matching posts, and *(ii)* the diversity dimension, e.g., the output will contain more posts from the time intervals with more posts.

The astute reader will notice that in contrast to the fixed λ setting, when λ is post-specific, the post coverage relation becomes directional. That is, it is possible that P_i λ-covers $a \in P_j$ but not P_j λ-covers $a \in P_i$. Nevertheless, all proposed algorithms can be easily adapted to incorporate this property. It does not fundamentally change the implementation of these algorithms except computing λ for each post per label. For Algorithm GreedySC and Algorithm Scan, it is straightforward to apply post and label-specific λs. For Algorithm OPT, one detail is that when processing post P_j we need to generate j-end-patterns and hence we need to find all the posts P_i λ-covers $a \in P_j$, where $|t_i - t_j|$ may be larger than $\lambda_a(P_j)$ as there might be $\lambda_a(P_i) > \lambda_a(P_j)$ s.t. P_i λ-covers $a \in P_j$ but not P_j λ-covers $a \in P_i$. This could potentially reduce the efficiency of OPT.

7. EXPERIMENTAL EVALUATION

In this section we study the effectiveness and efficiency of the proposed algorithms for MQDP and StreamMQDP. We describe the experimental setting in Section 7.1. Sections 7.2 and 7.3 study the effectiveness and efficiency of the algorithms, respectively.

7.1 Experimental Setting

Datasets. We conduct our experiments on Twitter data, that is, each document is a tweet. We use topic modeling to extract a set of topics, which we use as queries (labels), that is, each topic is mapped to a query,

Posts dataset. We used the Twitter Streaming API through which we can collect a random sample of up to 1% of the whole public Twitter stream. We ran the streaming API for 24 hours, on June 12th, 2013, and collected about 4.3 million tweets.

Queries. Recall that a key motivation of our work is to monitor posts related to a user profile, represented as a set of keyword queries. Given the lack of public profile datasets, and the fact that a large ratio of tweets are commenting or referring to news articles [16], we generate a query set by viewing each news topic as a query.

We use a news articles collection to extract topics, instead of using the tweets dataset, because we expect that the topics quality will be higher and also we want to avoid any bias to the algorithms from selecting queries directly based on the documents dataset. In particular, we collected news articles from several popular news websites, such as CNN, BBC, NY Times, LA Times etc., through their RSS feeds during the first half year in 2013 (until Jun 15th). This news collection consists of over 1 million articles. We applied unsupervised Latent Dirichlet Allocation (LDA) using an open source implementation by Mallet[1] to generate 300 topics on this news collection (the number of topics is an input parameter). Each trained topic is a set of keywords with corresponding weights. We keep the top 40 highest-weight keywords for each topic.

To generate label sets L (user profiles), we assume that each user is interested in a broad topic like politics or sports, and specifies queries inside this broad topic. The 300 extracted topics are grouped into 10 broad topics by three researchers in our lab (if some researcher thought a topic was too ambiguous we discarded the topic, thus we have 215 topics left). Then, to generate a label set L, we first randomly pick a broad topic and then randomly pick $|\mathsf{L}|$ topics within the broad topic. Table 1 shows some example topics.

[1]http://mallet.cs.umass.edu/

Topic	Keywords
Sports	woods tiger golf masters championship mcilroy garcia pga augusta rory mickelson
	nfl super bowl blog draft ravens ers football baltimore patriots jets quarterback giants eagles
Politics	obama president barack michelle inauguration house administration congress presidential republicans
	election vote poll presidential party president political race candidate campaign electoral coalition

Table 1: Example topics with their highest weight keywords.

$\lvert\mathsf{L}\rvert$	number of posts
2	136
5	308
20	1180

Table 2: Number of matching posts per minute, for a label set for various label set sizes.

We create label sets with different sizes ($|\mathsf{L}|$). For each $|\mathsf{L}|$, we create 100 label sets. Table 2 shows the average number of unique tweets matching at least one label set per minute, where matching is defined as containing at least one keyword of the topic (label).

Implementation and Platform. The tweets inverted index shown in Figure 1 was implemented using Apache Lucene[2]. Other real-time indexing systems are also possible, such as EarlyBird [5] or LSII [25], although indexing is out of the scope of this paper. Note that an index is only used for the MQDP and not for the streaming problem variants. We have implemented all algorithms in Java, and we conducted our experiment on a Windows 7 machine with Intel i5 3.0GHz CPU and 8 GB RAM.

7.2 Effectiveness Study

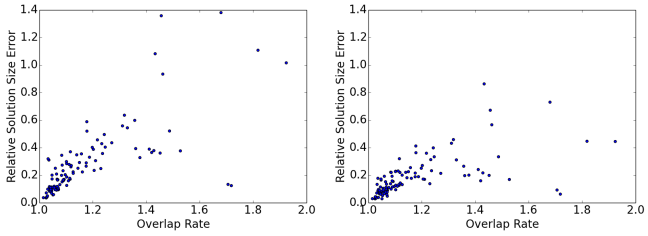
MQDP. Given that our exact dynamic programming algorithm OPT can only be executed on small problem instances, when evaluating the error of approximation algorithms, we only use a 10 minutes subset of our Twitter dataset, which starts at 12pm on Jun 13, and use small values for λ and $|\mathsf{L}|$, such that the number of j-end-patterns in OPT is not excessively large.

A key factor that may affect the effectiveness of the approximation algorithms is the overlap among tweets with respect to the labels of a label set. That is, if many tweets match multiple labels, then the problem is more challenging and hence the algorithms may have higher error bounds. We define the *post overlap rate* as the average number of labels a post is related to. Figure 6 shows the relative solution size error ($|estimated - optimal|/optimal$) of the approximation algorithms for various post overlap rates, for $|\mathsf{L}| = 3$. Each point in Figures 6a, 6b and 6c represents a label set. GreedySC generally has better (smaller) error than Scan and Scan+ except when the overlap rate is very close to 1; recall that Scan and Scan+ are optimal for a single label ($|\mathsf{L}| = 1$), and hence are also optimal for multiple labels if the posts have no overlap (no post is related to multiple labels); this is not the case for GreedySC. Figure 6d shows that the solution sizes in all algorithms generally drops when post overlap rate increases, as they can pick posts that cover posts matching multiple labels.

Figure 7 depicts the relative solution size error of the approximation algorithms for various λ values. We see that all approximation algorithms have higher error with larger λ values, because there are more possible choices and hence the problem becomes harder.

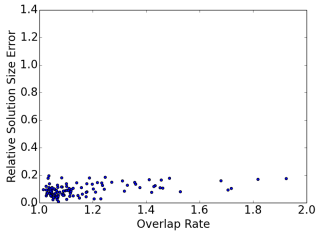
Figure 8 shows the solution sizes of the approximation algorithms on larger instances for varying number of labels, using the

[2]http://lucene.apache.org/

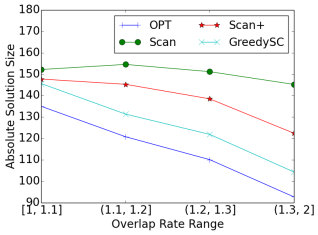


(a) Scan

(b) Scan+



(c) GreedySC



(d) Absolute solution size

Figure 6: Solution size errors and absolute solution sizes for $|\mathsf{L}| = 3$ and λ=5 seconds on a 10 minute interval, for varying overlap.

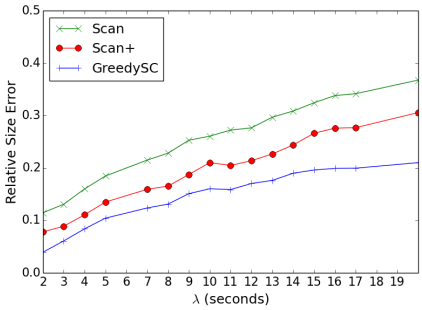
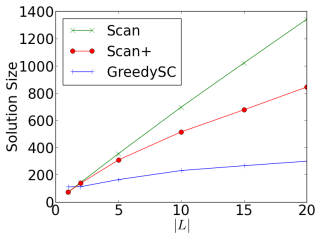
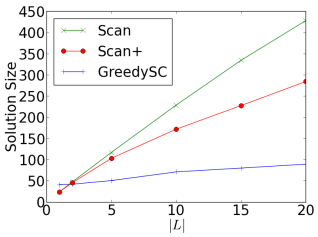


Figure 7: Relative solution size error for $|\mathsf{L}| = 2$ for varying λ on a 10 minute interval.

whole 1-day dataset. We see that the solution size of Scan is linear on $|\mathsf{L}|$ since it handles each label separately. GreedySC outperforms the other algorithms, especially as $|\mathsf{L}|$ increases.



(a) $\lambda = 10$ minutes



(b) $\lambda = 30$ minutes

Figure 8: Solution sizes on 1 day of tweets, for various label set sizes ($|\mathsf{L}|$).

StreamMQDP. In the streaming setting, it is tricky to define what the optimal solution is because an algorithm has to make a decision about outputting a post before knowing what will follow. To avoid this confusion, we consider as optimal the solution of an optimal algorithm that has full knowledge of the future posts. That is, the optimal streaming solution for a time interval is the same as the optimal static solution for the same interval. We again use a 10-minute time interval when the optimal solution is required.

Figure 9 presents the relative error for various λ values, given a fixed decision delay τ. We see that the relative errors generally increase as λ increases, since more coverage combinations are possible and hence the problem is harder; this is consistent with the analysis we did for MQDP. We also see that StreamGreedySC+ is consistently slightly better than StreamGreedySC.

Figure 10 shows the relative error for different τ values, given a fixed λ. We see that the Scan-based algorithms have stable error when $\tau > \lambda$ because then the streaming Scan algorithms generate the same solution as their non-streaming counterparts as discussed in Section 5.1.

A surprising and interesting observation for the greedy algorithms in both Figures 9 and 10 is that the error has a local peak when τ is slightly larger than 2λ and the smallest error is achieved when $\lambda = \tau$. We can explain the behavior based on the "in-between" posts, that is, short ($<< \lambda$) ranges of uncovered posts that are between already covered ranges, and to cover them, we incur big overlap with what is already covered. Hence: (a) We have a minimum at $\tau = \lambda$ because we are making sure that there are no "in-between" posts. (b) When $\tau >= \lambda$ we have a maximum at τ slightly bigger than 2λ because this maximizes the effect of "in-between" posts. The algorithm has a relatively high probability of using two (or three) posts to cover the posts of a label in this τ interval.

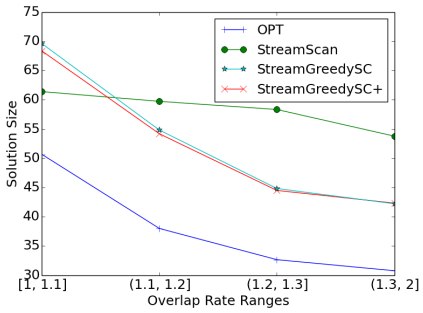


Figure 11: Absolute solution size when $|\mathsf{L}| = 2$ for a 10-minute interval for various overlap rate ranges.

Figure 11 shows the effect of the overlap rate on the solution sizes for $\lambda = 10$ seconds and $\tau = 5$ seconds. We see that the approximation algorithms follow the same trend as their static versions, that is, the greedy algorithms are better for higher overlap, whereas the Scan algorithms are better for small overlap (recall that Scan is optimal for overlap = 1).

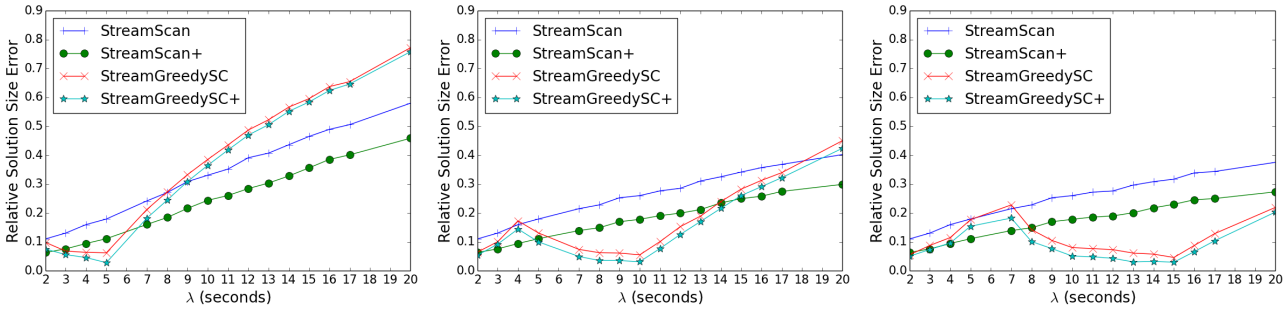
Similarly to Figure 8 for MQDP, Figure 12 shows the solution sizes of the approximation algorithms for StreamMQDP on one day of tweets. It shows that StreamGreedySC is better than StreamGreedySC+ on large λ.

7.3 Efficiency Study

We conduct our experiments on the one-day dataset. We measure the execution time of the algorithms on in-memory data, that is, we do not account for the I/O of loading the inverted indexes into memory.

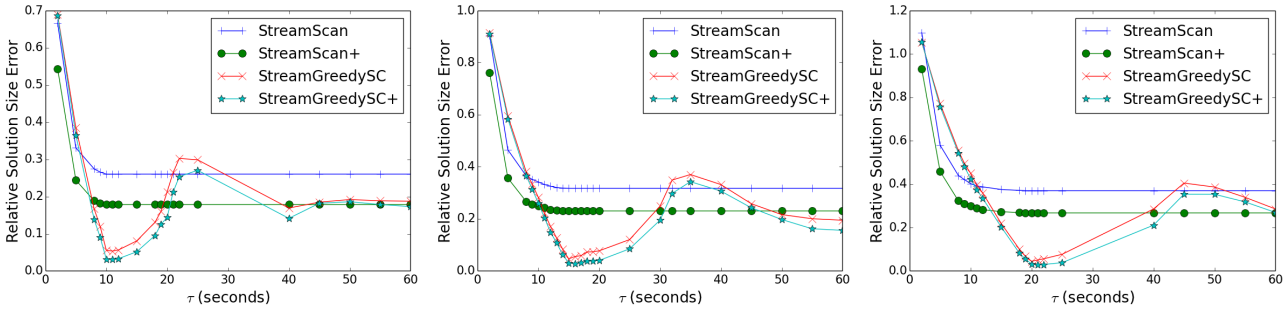
Since different queries (labels) may return quite different number of relevant posts, we compute the *execution time per post*, which is what is important to understand the throughput of posts that our algorithms can handle. We measure the execution times of each algorithm for $|\mathsf{L}| = 2$, 5, and 20.

MQDP. Figure 13 shows the efficiency results for varying λ on logarithmic axis. We generally see that Scan algorithms are orders of magnitude faster than the greedy ones, since they only make a



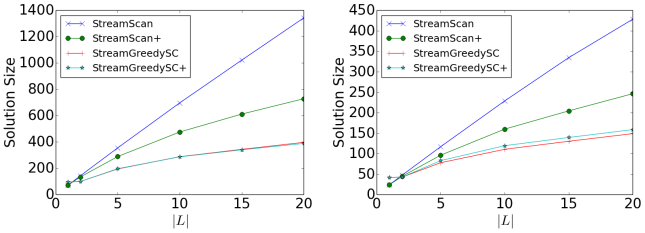
(a) $\tau = 5$ seconds (b) $\tau = 10$ seconds (c) $\tau = 15$ seconds

Figure 9: Relative solution size errors on 10-minute interval for varying λ when $|\mathsf{L}| = 2$.



(a) $\lambda = 10$ seconds (b) $\lambda = 15$ seconds (c) $\lambda = 20$ seconds

Figure 10: Relative solution size errors from approximation algorithms with respect to τ when $|\mathsf{L}| = 2$.



(a) $\lambda = 10$ minutes (b) $\lambda = 30$ minutes

Figure 12: Solution sizes of approximation algorithms for 1-day of posts for various $|\mathsf{L}|$, for $\tau = 30$ seconds.

sequential pass on the data. The running time of Scan and Scan+ is quite stable for different λs, whereas the efficiency of GreedySC increases sharply when λ increases, because the solution size is smaller with larger λ and hence GreedySC will have smaller number of rounds to pick posts and update the covered range of left posts.

Another observation is that for larger label set size $|\mathsf{L}|$, the running time of Scan and Scan+ decreases, since the same post may cover more other posts. However, GreedySC becomes slower with larger $|\mathsf{L}|$, since we need to execute more iterations to find the post with maximum cover range and to update other posts' cover ranges.

To better understand this behavior, we provide more details on our implementation of the greedy algorithms. We could maintain a heap (PriorityQueue in Java) in order to find the set with the maximum size at each step. Every time we select a post (a set in the set cover problem instance), we need to update other posts' cover range. For this, we need to delete the original set from the heap and insert the set back with updated weight (set size). When there are lots of relevant posts in a short time, this leads to big overhead of updating this heap. Thus we don't apply this in our implementation. Instead, we iterate all sets to find the set with maximum size, which is experimentally shown to have better performance on our data set.

StreamMQDP. Figure 14 evaluates the execution time of the approximation algorithms by varying λ with a fixed τ value, and Figure 15 shows the results by varying τ given a λ. Like Scan and Scan+ on MQDP, the efficiency of StreamScan and StreamScan+ for StreamMQDP is stable with respect to λ and τ. Given a λ, the efficiency of StreamGreedySC and StreamGreedySC+ decreases slightly with the increases of τ, and given a τ their execution time generally decreases with larger λ, because of smaller number of iterations of execution of set cover algorithm.

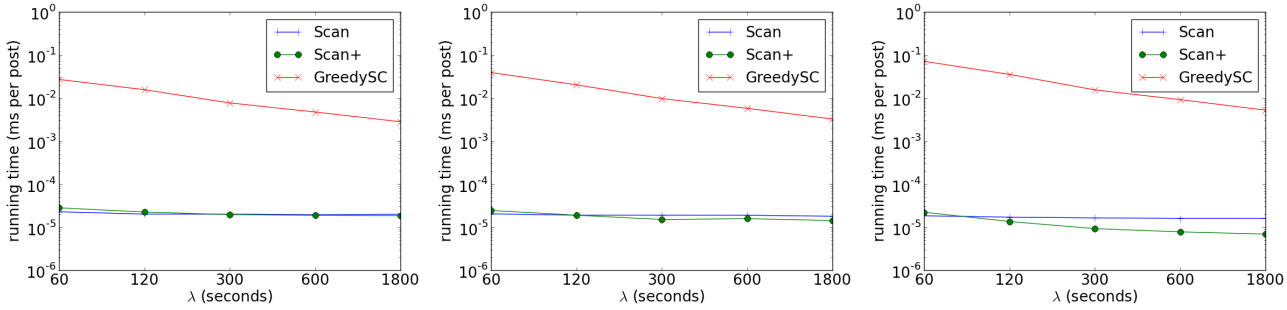
7.4 Discussion

Our experiments show that the two classes of presented approximate algorithms, Scan and GreedySC (including their variants), offer good relative errors ranging from 0.01 to 0.5 (except the extremely small τs in the streaming setting).

GreedySC has lower error than Scan in most settings; its maximum improvement is about 60% (see difference between GreedySC and Scan+ for $\lambda = 20$ seconds in Figure 7) for the static problem setting, whereas in the streaming setting GreedySC's error is unstable, and is often larger than that of Scan (see Figures 9 and 10).

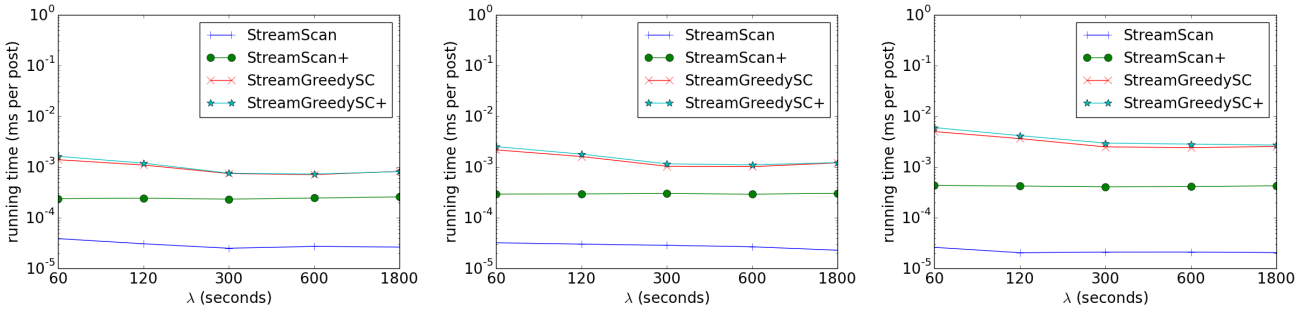
In addition to the instability, a potentially more serious shortcoming of GreedySC is that it is 1 to 3 orders of magnitude slower than Scan, which makes Scan more desirable for setting of high throughput and especially when the algorithm has to be executed for millions of users (as in Twitter).

Finally, we show that our proposed exact dynamic programming algorithm is feasible for small problem instances, where the number of queries is up to 2-3 and λ is less than a minute.



(a) $|\mathsf{L}| = 2$ (b) $|\mathsf{L}| = 5$ (c) $|\mathsf{L}| = 20$

Figure 13: Execution time for MQDP on one day of tweets, for varying λ.



(a) $|\mathsf{L}| = 2$ (b) $|\mathsf{L}| = 5$ (c) $|\mathsf{L}| = 20$

Figure 14: Execution time for StreamMQDP on one day of tweets, for varying λ with fixed $\tau = 300$ seconds.

8. RELATED WORK

Vertical and federal search. Previous works in aggregated search argue that a user's intent can be captured in a better way if the web search retrieves and integrates results from different vertical domains. Verticals can include for example microblogging sites (such as Twitter or Google+), user comments, blog feeds or breaking news. Additional information found in those verticals can be used in various ways.

Diaz [8] proposes principled ways to integrate news content into web search results. In [9] the authors rerank the web documents based on a set of social network features, such as the number of tweets that refer to each document, the number of retweets of these posts, etc. Similarly, [23] applies a vector space model technique in order to rerank web results based on their similarity with tweet posts published in a user's social circle. Our approach in this paper is different since we do not aim to merge microblogging posts with web search results.

Linking microblogging posts with news articles. With a similar motivation to our paper, several works have focused on the relation between news articles and microblogging posts such as those commenting on, referring to, or directly related through links to a news article or specific event [14, 21, 24, 1, 13, 15]. These works can be considered complementary to our approach. That is, instead of treating a news article as a query to retrieve related microblogging posts, we can apply these works in order to build the relation between articles and posts. However, efficiency could be a potential issue in such a scenario.

Diversity. Query results diversification is a well-studied problem in the field of information retrieval [2, 20] as well as in data management [19, 10]. Because of the ambiguity of queries, results diversification is very helpful to satisfy the search intent of different users. In this paper, we take a different definition of diversification from these works, which is based on multiple queries. Further, similarly to [10], we put more focus on results coverage. Santos et al. [22] work on diversification on explicit sub-queries of the original query. They maximize the semantic coverage with respect to different aspects of the original query.

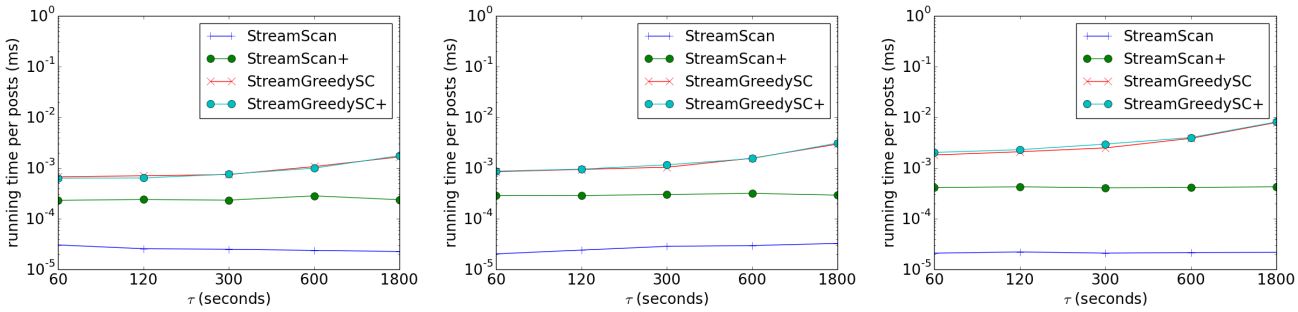
Giannopoulos et al. [13] address the problem of diversifying user comments found on news articles such that the selected comments cover different aspects of the article. Compared to [13], our work focuses on identifying microblogging posts that refer to multiple news articles, whereas [13] provides a solution only for one article. Further, instead of maximizing diversity, we use a coverage-based optimization goal. Finally, we mainly focus on the efficiency of the proposed methods, whereas [13] do not consider this aspect.

Publish/subscribe. Different variations of *Publish/Subscribe* systems have been proposed such as Topic-Based, Content-Based, and Type-Based [11]. However, to the best of our knowledge, there has not been considerable work on building publish/subscribe systems on microblogging services. The authors in [7] suggest a publish/subscribe infrastructure for microblogging services to better support crowdsourced sensing and collaboration applications.

9. CONCLUSIONS AND FUTURE WORK

In this paper we introduced and formalized the Multi-Query Diversification Problem (MQDP). We proved MQDP is NP-hard and we proposed several algorithms for solving it: an exact dynamic programming algorithm, two efficient approximation algorithms, as well as some algorithms for the streaming variant. We confirmed the effectiveness of our approach through extensive experiments on real Twitter data set.

In the future we will study how our solutions can be adapted for more diversity dimensions. In particular, we would like to extend them to the spatiotemporal space, where the selected posts need to cover both the time and geospatial dimension. Incorporating geographical information is likely to become more important as,



(a) $|\mathsf{L}| = 2$ (b) $|\mathsf{L}| = 5$ (c) $|\mathsf{L}| = 20$

Figure 15: Execution time for StreamMQDP on one day of tweets, for varying τ with fixed $\lambda = 300$ seconds.

increasingly, more posts are geotagged.

Further, we will study how content-based intra-result diversity methods [2, 3, 19, 10] can be effectively applied to short posts [18] – for instance through context expansion – and then how this content-based diversity can be represented in our multi-query diversity framework.

10. ACKNOWLEDGMENTS

Marek Chrobak is supported by National Science Foundation grant CCF-1217314. Vagelis Hristidis is supported by National Science Foundation grant IIS-1216007 and DTRA grant HDTRA1-13-C-0078.

11. REFERENCES

[1] F. Abel, Q. Gao, G.-J. Houben, and K. Tao. Semantic enrichment of twitter posts for user profile construction on the social web. In *ESWC*, pages 375–389, 2011.

[2] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM*, pages 5–14, 2009.

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

[4] N. J. Belkin, P. B. Kantor, E. A. Fox, and J. A. Shaw. Combining the evidence of multiple query representations for information retrieval. *Inf. Process. Manage.*, 31(3):431–448, 1995.

[5] M. Busch, K. Gade, B. Larson, P. Lok, S. Luckenbill, and J. Lin. Earlybird: Real-time search at twitter. In *ICDE*, pages 1360–1369, 2012.

[6] C. Chen, F. Li, B. C. Ooi, and S. Wu. TI: an efficient indexing mechanism for real-time search on tweets. In *SIGMOD*, pages 649–660, 2011.

[7] M. Demirbas, M. A. Bayir, C. G. Akcora, Y. S. Yilmaz, and H. Ferhatosmanoglu. Crowd-sourced sensing and collaboration using twitter. In *WOWMOM*, pages 1–9, 2010.

[8] F. Diaz. Integration of news content into web results. In *WSDM*, pages 182–191, 2009.

[9] A. Dong, R. Zhang, P. Kolari, J. Bai, F. Diaz, Y. Chang, Z. Zheng, and H. Zha. Time is of the essence: improving recency ranking using Twitter data. In *WWW*, pages 331–340, 2010.

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

[11] P. T. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[12] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[13] G. Giannopoulos, I. Weber, A. Jaimes, and T. K. Sellis. Diversifying user comments on news articles. In *WISE*, pages 100–113, 2012.

[14] D. Ikeda, T. Fujiki, and M. Okumura. Automatically linking news articles to blog entries. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*, pages 78–82, 2006.

[15] A. Kothari, W. Magdy, A. M. Kareem Darwish, and A. Taei. Detecting comments on news articles in microblogs. *ICWSM*, 2013.

[16] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *WWW*, pages 591–600, 2010.

[17] G. S. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. In *WWW*, pages 141–150. ACM, 2007.

[18] E. Meij, W. Weerkamp, and M. de Rijke. Adding semantics to microblog posts. In *WSDM*, pages 563–572. ACM, 2012.

[19] L. Qin, J. X. Yu, and L. Chang. Diversifying top-k results. *PVLDB*, 5(11):1124–1135, 2012.

[20] D. Rafiei, K. Bharat, and A. Shukla. Diversifying web search results. In *WWW*, pages 781–790, 2010.

[21] J. Sankaranarayanan, H. Samet, B. E. Teitler, M. D. Lieberman, and J. Sperling. Twitterstand: news in tweets. In *GIS*, pages 42–51, 2009.

[22] R. L. T. Santos, J. Peng, C. Macdonald, and I. Ounis. Explicit search result diversification through sub-queries. In *ECIR*, pages 87–99, 2010.

[23] X. Shuai, X. Liu, and J. Bollen. Improving news ranking by community tweets. In *WWW*, pages 1227–1232, 2012.

[24] M. Tsagkias, M. de Rijke, and W. Weerkamp. Linking online news and social media. In *WSDM*, pages 565–574, 2011.

[25] L. Wu, W. Lin, X. Xiao, and Y. Xu. LSII: An indexing structure for exact real-time search on microblogs. In *ICDE*, 2013.

[26] L. Wu, Y. Wang, J. Shepherd, and X. Zhao. An optimization method for proportionally diversifying search results. In *Advances in Knowledge Discovery and Data Mining*, pages 390–401. 2013.