Slowing the Firehose: Multi-Dimensional Diversity on Social Post Streams

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

ABSTRACT

Web 2.0 users conveniently consume content through subscribing to content generators such as Twitter users or news agencies. However, given the number of subscriptions and the rate of the subscription streams, users suffer from the information overload problem. To address this issue, we propose a novel and flexible diversification paradigm to prune redundant posts from a collection of streams. A key novelty of our diversification model is that it holistically incorporates three important dimensions of social posts, namely content, time and author. We show how different applications, such as microblogging, news or bibliographic services, require different settings for these three dimensions. Further, each dimension poses unique performance challenges towards scaling the diversification model for many users and many high-throughput streams. We show that hash-based content distance measures and graph-based author distance measures are both effective and efficient for social posts. We propose scalable real-time stream processing algorithms leveraging efficient indexes that input a social post stream and output a diversified version of the stream, diversified across all three dimensions. Next, we show how these techniques can be extended to serve multiple users by appropriately reusing indexing and computation where possible. Through extensive experiments on real Twitter data, we show that our diversification model is effective and our solutions are scalable. We show that different algorithms perform best for different application settings.

1. INTRODUCTION

Tremendous amounts of online social data are generated every day. For instance, Twitter has reported over 280 million monthly active users in its microblogging service and 500 million Tweets posted per day[1]. One common way to consume social data is through implicit or explicit subscription. For example, almost all news agencies offer RSS feeds for people to subscribe. Google Scholar continuously recommends new publications to its users based on a user's profile and publication history. In a microblogging system like Twitter, one can subscribe to other users' posts by following them.

[1]https://about.twitter.com/company

All posts matching a user's subscriptions are typically displayed in a convenient central place, such as the user's timeline in Twitter or Facebook. These timelines are updated in real time. A key challenge is that a user could be easily overwhelmed by the number of posts in the timeline, especially if the user is subscribed to many post producers. Further, a user's timeline often contains lots of posts that carry no new information with respect to other similar posts. This data overload issue also happens in other applications with smaller data throughput such as news and research papers. For instance, it has been shown that a primary care physician should read hundreds of medical publications per day to keep up with the medical literature [2].

To alleviate the data overload problem, in this paper we propose a novel way to efficiently and effectively diversify social post streams by pruning redundant posts. By social post streams we mean a broad class of content generated by services where each post, in addition to its textual content, has a unique author and a unique timestamp, and where authors are associated through various social relationships. For instance, in Google Scholar authors are connected by relations such as co-authorship or overlapping research interests. In microblogging sites users are connected by follower/followee relations.

Given a stream consisting of all the posts from a user's subscriptions, our goal is to output in real-time a subset of the stream in which *(i)* all posts are dissimilar to each other and *(ii)* any post in the whole stream will be either included or *covered* by a post in the sub-stream. A post covers another post if the two posts are similar in all three similarity dimensions: (a) content, (b) time and (c) author.

Two posts have similar *content* if their text components are similar. Intuitively, all other dimensions being equal, users want to avoid seeing two posts with very similar content. Similarly, the *timestamp* distance of two posts is important in social post diversification. Two posts that have similar content but are far away in terms of post time, may both be of interest to the user. Note that time is widely used for diversifying search results in microblogging systems [10, 14, 4].

The *author* similarity is a more subtle dimension that to the best of our knowledge has not been used before for computing diversity in social media. For example, CNN and Fox News, which both have official Twitter accounts, are dissimilar to each other because they generally have different political views. We compute the distance between two authors through their social connections. In particular, we compare the sets of friends (or followers in the case of Twitter) of the two authors, which has been shown to be a good author similarity measure in social networks [21, 9].

Challenges: To summarize, in our model two posts are redundant with respect to each other if they are similar in all of the three

dimensions. It is challenging to apply the proposed diversification model in a large scale social service with high posts throughput. First, we must efficiently compare the content of a new post to the content of all previous posts (within a time window). For this, we apply Hash-based techniques to measure the content similarity between social posts. Hash-based techniques have been applied before to Web documents [11], but not to social posts, which are generally shorter and may heavily rely on abbreviations or URLs.

Second, handling the author dimension is challenging. A naive approach is to check if the author of each new post is similar to the author of each existing post (within a time window). However, we show that depending on the setting (similarity thresholds across the three dimensions), a different indexing data structure is more efficient to achieve real-time posts processing.

Third, the three diversity dimensions offer an opportunity to use the results of the one dimension to prune the work needed for the other dimension. For instance, if a reader knows that posts P_1 and P_2 have high content similarity, then she doesn't need to check if their authors or time are similar.

Fourth, if we move from one user to many users, where each user has a collection of subscriptions, the challenge is how to reuse the computation performed for diversifying one user's stream to diversify streams of other users. We show that we can reuse computation across users only if their shared subscriptions meet a strict condition.

Previous work on diversity: There has been much work on diversifying results for documents [15, 1, 3], social posts [10, 14, 4] and database records [5, 6]. However, none of these works can be applied to our setting where: (i) data is streaming and an instant decision must be made on whether a post should be pushed to the user, and (ii) a multi-dimensional diversity model is adopted. In contrast, most previous works focus on the search setting, where a user submits a query and the set of results must be diversified based on content, including work on social posts [10, 14].

The problem studied in this paper is also fundamentally different from previous work on stream summarization [20, 18, 16, 23], because: *(i)* we do not aim to generate an aggregation of documents, but instead select a subset of posts, and *(ii)* we define strict coverage constraints to guarantee that not even one uncovered posts is missed.

Contributions: In this paper, we make following contributions:

- We propose a new paradigm to define diversity on social posts, by incorporating three important dimensions – content, time and author – and we define corresponding optimization problems (Section 2).
- We study how content similarity can be efficiently applied to social posts, which are generally short and contain abbreviations (Section 3).
- We propose efficient data structures and algorithms to solve the social posts stream diversification problem (Section 4).
- We show how the single-user algorithm can be extended to handle many users, by reusing computation across users (Section 5).
- We perform a comprehensive experimental evaluation, where we focus on microblogging data, which poses the most serious scalability challenges. We show how different algorithms perform better for different diversity needs (Section 6).

Section 7 reviews related work. We conclude in Section 8.

2. FRAMEWORK AND PROBLEM DEFINITION

Let P represent a stream (ordered set) of social posts. Each post P_i in P has an author $author(P_i)$, textual content $text(P_i)$ and a timestamp $time(P_i)$ (also referred as t_i). We define the distance measures across the three diversity dimensions as follows.

- **Content Distance.** We represent the content distance between two posts P_i and P_j as $dist_c(P_i, P_j)$. Cosine similarity is a possible way to define the distance, but for efficiency purposes we employ the hash-based simhash measure as explained in Section 3, where we show that simhash is effective for social posts.
- **Time Distance.** The time distance between two posts P_i and P_j is denoted as $dist_t(P_i, P_j) = |t_i - t_j|$.
- **Author Distance.** We denote the author distance between P_i and P_j as $dist_a(P_i, P_j)$. For social data, we define the similarity between two authors as the cosine similarity between their friends' vectors, which has been successfully used in previous work to measure the user similarity in Twitter [21, 9]. The author distance is $(1 - similarity)$. For other domains other distance measures may be more appropriate.

Next, we define the coverage semantics between posts.

Definition 1. *(Post Coverage) Given a content diversity threshold λ_c, a time diversity threshold λ_t and an author diversity threshold λ_a, two social posts P_i and P_j cover each other if:*

- $dist_c(P_i, P_j) \leq \lambda_c$ *and*
- $dist_t(P_i, P_j) \leq \lambda_t$ *and*
- $dist_a(P_i, P_j) \leq \lambda_a$.

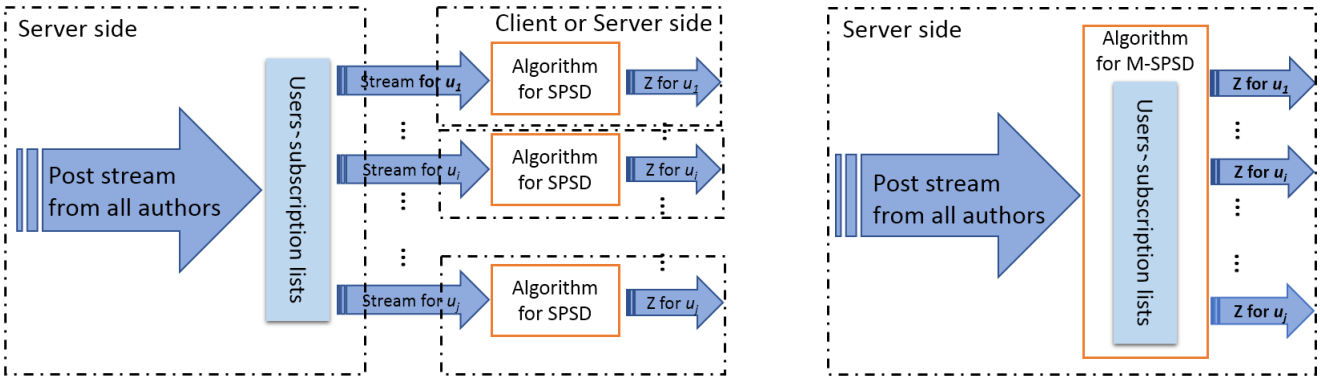
Note that the coverage semantics between two posts is symmetric. The three thresholds may vary according to the characteristics of a social system as we discuss below. The primary focus of this paper is to study the efficient processing of a posts stream and not to set these threshold values.

We next define the Social Post Stream Diversification (SPSD) problem.

Problem 1 [Social Post Stream Diversification (SPSD)] Given a social post stream P, and diversity thresholds λ_c, λ_t and λ_a, compute a sub-stream of posts $Z \subseteq \mathsf{P}$ that covers P, that is, $\forall P_i \in \mathsf{P}$ $\exists P_j \in Z$, such that P_j covers P_i.

Note we have to compute Z in *real-time*, i.e., immediately decide whether a post P_i should be included in Z at its arrival. That is, we cannot first view the whole stream and then decide which posts should be included in the substream.

In SPSD, there is a single user who consumes the stream and many authors who generate the posts of the stream (a user may also be an author and vice versa). That is, a solution to SPSD should be deployed for each user, for example, as part of the Twitter app of a user. On the other hand, a social network service would rather have a central diversification engine that diversifies the posts for each of its users, so that no client side post processing is required. We refer to this version of SPSD as Multiple-Users SPSD (M-SPSD). Another difference between SPSD and M-SPSD is that in SPSD we can easily support user customized diversity thresholds. Figure 1 shows how SPSD and M-SPSD differ in terms of the setting and deployment.



(a) SPSD (b) M-SPSD

Figure 1: Settings of SPDP and M-SPDP.

Problem 2 [Multiple-Users Social Post Stream Diversification (M-SPSD)] Given a social post stream P, diversity thresholds λ_c, λ_t and λ_a, and a set of users where each user is subscribed to a subset of the authors, compute a diversified sub-stream for each user.

3. CONTENT DISTANCE ESTIMATION FOR MICROBLOGGING POSTS

Among the three diversity dimensions, the content distance is the most expensive to compute, because it must be computed for each new post. This is especially true given our real-time decision semantics described above. In contrast, the author similarity between each pair of authors may be precomputed (e.g., once every week), as it changes slowly over time. For that reason, we cannot afford to use traditional content similarity measures such as cosine similarity. Instead, we turn to hash-based distance measures. In this section we present the details of the employed content distance technique along with an analysis of its effectiveness for microblogging data.

We define the content distance between two posts P_i and P_j as the Hamming distance of their SimHash [17] fingerprints. Previous work has applied SimHash on web documents [11] and showed that it is efficient and effective. We represent the SimHash of $text(P_i)$ as S_i, which is a 64-bit fingerprint. The Hamming distance of two SimHash fingerprints is the number of different bits between them. According to the experimental analysis in [19], the cosine distance between two texts positively correlates to the Hamming distance of their corresponding SimHash fingerprints.

Distribution of SimHash distances in Twitter

First, we study the distribution of SimHash distances on Twitter data. We collected a dataset of 200 thousand tweets from the Twitter Streaming API, which returns a stream of randomly selected substream of Twitter ([12] showed that the stream is not exactly random but this is not too important for our problem). The distribution of the Hamming distances for these tweets is depicted in Figure 2, which shows a perfect normal distribution with mean value 32, as expected, and with most of the distances between 24 to 40.

User Study

To further evaluate the effectiveness of SimHash for social posts, we conducted a user study to learn the relationship between the SimHash distance between two posts and the perceived dissimilarity between the posts. A second goal of the study is to learn what is a good SimHash distance threshold (e.g., a threshold of 3 bits was chosen to define redundant Web pages [11]) and if any preprocessing of the tweet text (e.g., expand shortened URLs) may improve the effectiveness of SimHash.

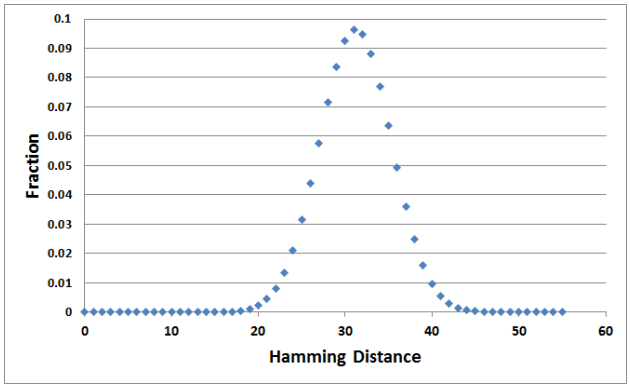


Figure 2: Hamming distance distribution

Setup and Methods: In particular, we collected a dataset of 2000 pairs of tweets randomly selected from the 200,000 tweets returned by the Twitter Streaming API, with SimHash distances between 3 and 22 – 100 tweets from each distance value. We chose 3 to 22 because this is the range where we expect to find posts that are very similar (redundant with respect to each other). This range choice is supported by our results below. We recruited 12 undergraduate and graduate students.

We evenly divided these 2000 pairs into 4 groups and distributed them to the 12 students for labeling. The author and timestamp of the posts are hidden. Some examples of these pairs are shown in Table 1. Each group of tweets is labeled by 3 students. The students were asked to mark whether the two tweets in a pair are redundant with respect to each other.

To help the users more accurately label the similarity between two posts, we showed the expanded URL (instead of the shortened one shown in Table 1). We used a majority vote, that is, if at least 2 out of the 3 students labelled a pair as redundant, we labelled the pair as near-duplicates.

Results: Out of the 2000 pairs, the users marked 949 pairs as redundant. Figure 3 shows the precision and recall achieved by various SimHash distance values. For each Hamming distance h, the precision is defined as the fraction of pairs with Hamming distance no more than h that are true near-duplicates. Recall is the

Table 1: Example tweet pairs and their Hamming distances

Tweet pair	*Hamming distance*
Over 300 people missing after South Korean ferry sinks. (Reuters) Story: http://t.co/9w2JrurhKm Over 300 people missing after South Korean ferry sinks. (Reuters) Story: http://t.co/E1vKp9JJfe	3
"In order to succeed, your desire for success should be greater than your fear of failure" Bill Cosby In order to succeed, your desire for success should be greater than your fear of failure. #quote #success - Bill Cosby	8
Alibaba's growth accelerates, U.S. IPO filing expected next week http://t.co/mUcmLJ4cpc #Technology #Reuters Alibaba's growth accelerates, U.S. IPO filing expected next week: SAN FRANCISCO (Reuters) - Alibaba Group Hold... http://t.co/aLAV8w4gWF	13

fraction of the total number of near-duplicate pairs that are detected with Hamming distance at most h. This graph shows that SimHash distance is an effective measure to identify similar posts.

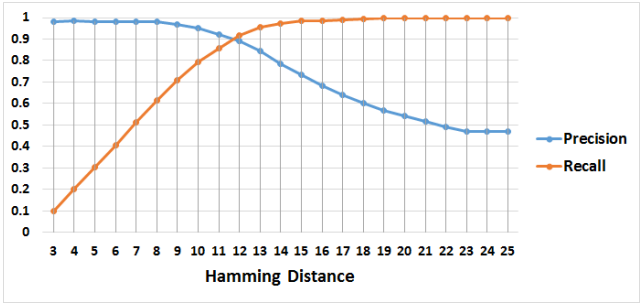


Figure 3: Precision and Recall for Hamming distance. SimHash fingerprints are generated from raw texts of tweets

Next, we study if various text preprocessing methods may improve the precision or recall of SimHash distance measure for microblogs. We first normalize the text by (a) changing all text to lowercase, (b) removing extra white spaces between words, and (c) removing non-alphanumeric characters (such as $*,, -, +, /$, etc.). Figure 4 plots the precision and recall after we apply the normalization. We see that this graph achieves higher precision and recall values than the original analysis in Figure 3. We also see that the the two lines cross for $distance = 18$, which achieves precision $= 0.96$ and recall $= 0.95$.Hence, we use $\lambda_c = 18$ as the default content distance threshold in the experiments in Section 6.

We also tried other methods of text preprocessing such as expanding shortened URLs (URLs in tweets are shortened by Twitter), varying the weights of user mentions and hashtags (by creating artificial copies), and expanding abbreviations. However, these methods had no significant impact to the precision and recall.

For completeness, we compared the effectiveness of SimHash to that of cosine similarity (which is much slower as discussed above) in terms of detecting posts with near-duplicate content (redundant). We tried different cosine threshold values and found that the precision and recall lines across at cosine similarity 0.7, where all posts with cosine similarity above 0.7 are marked as redundant. This achieves precision and recall of 0.96 and 0.95 respectively, which is the same as what we achieved using SimHash above. This means that, for detecting near-duplicate in our dataset, SimHash achieves effectiveness similar to cosine similarity. Hence, given the time performance advantage of SimHash, it is the best choice for our problem.

The high threshold value of $\lambda_c = 18$ for SimHash precludes the use of the efficient SimHash index proposed in [11] which relies on building several copies of the SimHash values table for several permutations of the bits, since the number of these copies is exponential in λ_c (which was only 3 in [11]). Hence, as we discuss in Section 4, other indexing and searching techniques are required.

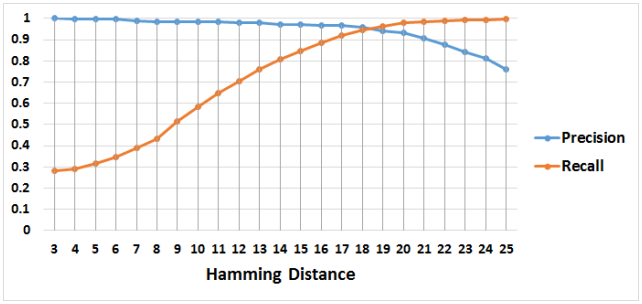


Figure 4: Precision and Recall for Hamming distance. SimHash fingerprints are generated from normalized texts of tweets

4. ALGORITHMS FOR SPSD

In this section, we describe our algorithmic solutions for the SPSD problem. As explained earlier in Section 3, due to the high Hamming distance threshold we are unable to use existing SimHash indexing techniques, and we must rely on comparing the SimHash value of each new post with those of all the previous ones, leading inevitably to linear time complexity per post in the worst case. We reduce the number of these comparisons by leveraging the other two dimensions, time and author. We first discuss how we handle time diversity, which is simpler, and then we present various approaches for handling author diversity.

Handling Time Diversity. According to the diversity model, at the arrival of a post P_i it can only be covered by the previous posts within a λ_t time distance. Thus, it is sufficient to store only the posts from previous λ_t time in memory for checking the coverage of a new post. One possible implementation is that we could store the posts in a circular array. We track two post indices for the oldest post within a λ_t distance to current time (a) and the most recent post (b). At the arrival of each post P_i, we compare it to the posts from most recent post to the oldest (i.e., from index b to a). If we encounter a post P_j with $t_i - t_j > \lambda_t$, we update a to be index of the post right after P_j. And we insert a non-redundant post to the array with index $(b + 1)$ and update $b = b + 1$.

Now that we have discussed how to handle time diversity, we focus on the author diversity among the posts in the last λ_t time units. The author similarity relations between all authors form an

author similarity graph G, which as we discussed above may be periodically precomputed. There is an edge between two authors in G if their distance is below the threshold λ_a. For each user u_i who subscribes to a set A of authors, we define G_i as the subgraph of G that contains all the A authors and the edges among them. In this section, we assume there is only one user (hence, one G_i) and in Section 5 we assume multiple users (and G_i's).

4.1 UniBin

Our first method to solve SPSD, which we refer as *UniBin*, works as follows: At the arrival of each post P_i in P, we sequentially (from the most recent post to the older ones) compare P_i to each post in the past λ_t time range in the diversified sub-stream Z. For each post P_j, we check whether P_j meets both: (1) Hamming distance between S_i and S_j (SimHash fingerprints of P_i and P_j, respectively) $\leq \lambda_c$, and (2) $dist_a(P_i, P_j) \leq \lambda_a$, which can be achieved by checking whether $author(P_i)$ and $author(P_j)$ are the same or neighbors in G. If no post from the past λ_t time range meets the above two conditions (i.e., P_i is not covered by Z), then we add P_i to Z. Otherwise we do not include P_i in Z.

We denote this method as **UniBin** indicating that the posts from all authors are stored in a single *post bin* (e.g., a circular array as described earlier). We illustrate UniBin with an example. In Figure 5a, each node represents an author. Two authors are connected by an edge if they are similar to each other (i.e., the author distance $\leq \lambda_a$). Figure 5b shows the posts from these authors with post distance information in terms of all three diversity dimensions.

We show the update of a post bin for UniBin in Figure 6a. When P_1 arrives, there is no posts in the bin yet. Thus P_1 is not covered hence is added to the bin. P_2 is also added as it is not covered by P_1 (the Hamming distance between S_1 and S_2, $dist_c(P_1, P_2)$, is larger than the threshold λ_c). For P_3, the algorithm first compares it to P_2 which does not cover P_3 (because $dist_c(P_2, P_3) > \lambda_c$). However, it is covered by P_1 because in all three diversity dimensions they are within the distance thresholds (or above similarity threshold). Thus, P_3 is not added. So forth, P_4 is not covered by either P_1 and P_2 and is included in the bin. However, we note that P_4 and P_3 cover each other. Finally, P_5 is covered by P_4.

4.2 NeighborBin

UniBin has to compare a new post (both its author and content SimHash) to all posts in the last λ_t time units. This aggregated time may be considerable given the high frequency of posts, even if the author similarity graph G_i and the post bin are maintained in memory.

To improve this, we partition the posts by their authors such that for a new post P_i we only check its coverage by comparing with the posts from $author(P_i)$ or from $author(P_i)$'s similar authors. Specifically, we create a post bin for each author.and when a new post P_i comes, the algorithm sequentially checks posts in the bin identified by $author(P_i)$ but not other posts. However, we must note that posts from the authors that are neighbors of $author(P_i)$ in G_i can potentially cover P_i. Hence, the post bin of an author also includes the posts of similar authors (neighbors in G_i). Thus, we add P_i to all bins of $author(P_i)$'s neighbors in addition to the bin of $author(P_i)$, if P_i is detected as a non-redundant post. We denote this method as **NeighborBin**.

Figure 6b depicts the execution of NeighborBin for the data shown in Figure 5. P_1 is added not only to the bin of its author a1, but also to the bins of a2 and a3, because they are neighbors of a1, as shown in Figure 5a. To check the coverage of P_2, only the post bin of a2 is accessed where P_1 does not cover P_2. After that, P_2 is also added to the post bins of a1, a2 and a3. NeighborBin checks the coverage of P_3 by iterating posts in the bin of a3 where P_1 covers P_3. When P_4 comes, a4's post bin is blank and thus P_4 is added to the post bins of a3 and a4 without incurring any post comparisons. Finally, P_5 is detected as redundant by checking the bin of $a3$ ($author(P_5)$ = a3) where P_4 covers P_5.

4.3 CliqueBin

In NeighborBin, we index the posts by author aiming to reduce the pairwise post comparisons. But the tradeoff is memory consumption: we have multiple copies of a post in different authors' post bins.

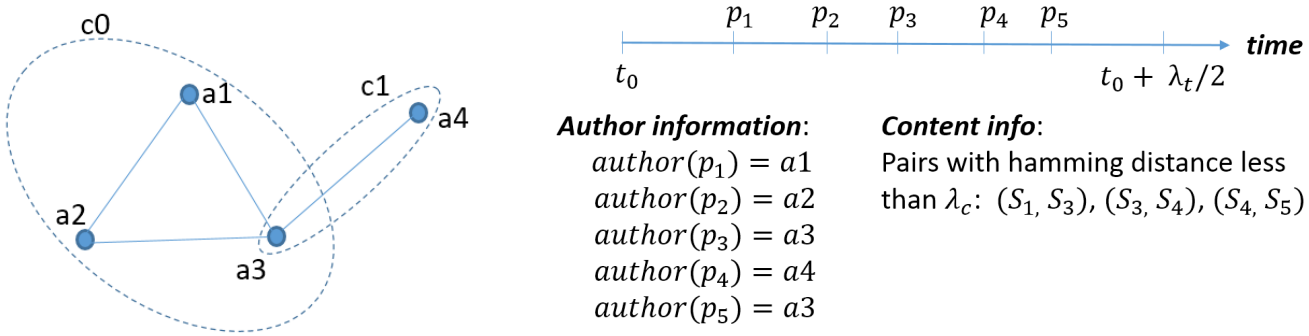
To reduce the overhead on memory consumption incurred by NeighborBin, we identify groups (cliques) of authors that are similar to each other and assign a single bin to them, such that a post generated by any of these authors is only stored in that bin. Specifically we find a *clique edge cover* of G_i, that is a collection of cliques whose union contains all edges of G_i. We maintain a post bin per clique (e.g., a map from clique ID to a list of posts). Only the posts from authors in a same clique as $author(P_i)$ can possibly cover post P_i. Thus, at the arrival of post P_i, we check whether it is covered by sequentially comparing it to the posts from only the cliques that contain $author(P_i)$. Thus a post P_i in Z is stored once for every clique that contains $author(P_i)$ – instead of once for each neighbor of $author(P_i)$ in NeighborBin. Note that this approach guarantees that the coverage requirement for posts is satisfied: when a new post P_i authored by a_j appears, and P_i is not similar to earlier posts of a_j or its neighbors then P_i will be added to the cliques involving a_j, because a_j's edges are covered by the cliques.

Considering the space consumption, our objective should be to minimize the sum of the sizes of cliques, i.e., the average number of cliques per author is minimized and thus number of copies per post is reduced. This is an NP-hard problem, and hence we have decided to use a simple greedy heuristic. It starts by picking an edge in G_i to form an initial clique. Then it extends the clique by adding nodes that are neighbors to all the nodes in the clique. When there is no such node, the clique is saved and the algorithm picks another edge not yet included in any found cliques and repeats the above process. We stop when all edges are covered.

Upon a new post P_i, we use a hashmap (Author2Cliques) to get all the cliques that contains $author(P_i)$, and then we check the posts in the corresponding bins. Recall that NeighborBin and UniBin load the author similarity graph G_i in memory. We can make the same assumption that Author2Cliques is loaded in memory for applying CliqueBin. Similar to the computation of author similarity graph, we assume the clique partition of G_i and the Author2Cliques mapping are computed offline. We denote this algorithm as **CliqueBin**.

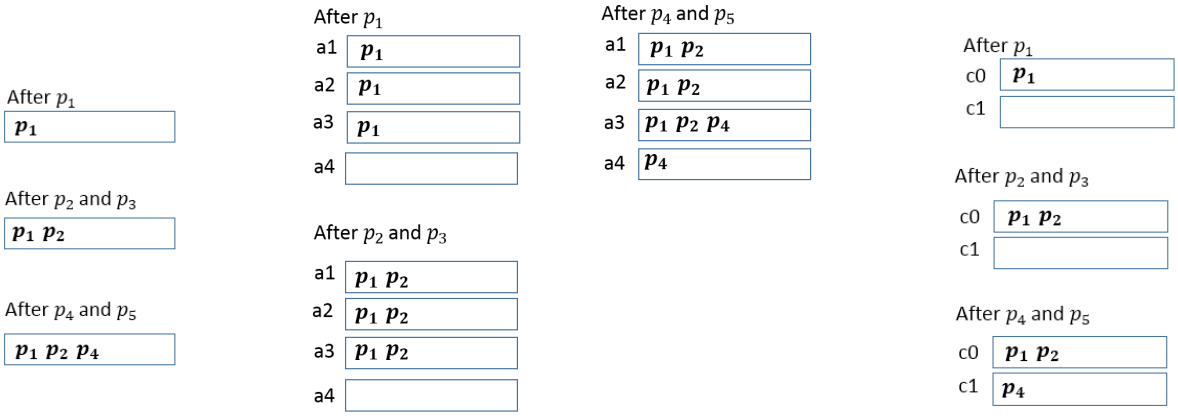
The update of a post bin by CliqueBin is depicted in Figure 6c. Cliques C0 and C1 together cover all the edges in the graph. We can see that P_1 is only stored once in C0's bin (because a1 is in C0) instead of saving 3 copies in NeighborBin as Figure 6b. The same applies to P_2. Since a3 is in both C0 and C1, during the processing of P_3 CliqueBin may check both bins of C0 and C1. P_4 will only be compared with the bin of C1 because a4 belongs to only C1. Again, CliqueBin checks the coverage of P_5 by iterating both bins of C0 and C1. This example illustrates how CliqueBin can reduce space requirements compared to NeighborBin.

We note that in some cases CliqueBin may have to do a larger number of pairwise post comparisons than NeighborBin. Suppose that after P_5 in the above example author a3 posts P_6 and then author a4 posts P_7. If P_6 and P_7 are not redundant to any other posts, then P_6 should be added to all four post bins in NeighborBin, and



(a) Author similarity graph G_i (b) Social Posts

Figure 5: Example of author similarity graph and posts



(a) UniBin (b) NeighborBin (c) CliqueBin

Figure 6: Running example for the three algorithms for SPSD.

to both post bins in CliqueBin. For P_7, NeighborBin only accesses the bin of a4 and thus only needs to do two comparisons (with P_4 and P_6). In contrast, CliqueBin has to do 5 comparisons: with P_1, P_2, P_4 and twice with P_6 (once in post bin of each clique). We study this experimentally in Section 6.

4.4 Performance Analysis

In this section we show an estimate of the time and space complexity of our algorithms, attempting to capture their performance on realistic data, rather than the worst-case performance. Rigorous derivation of such estimates is challenging, because the behavior of these algorithms heavily depends on the specifics of the data sets, including the topology of the social network. Instead, we provide informal derivations based on several reasonable assumptions about the data set and the graph's topology.

Suppose there are m subscribed authors, and the total number of posts from these m authors in a λ_t time range is n. We assume a ratio of r (≤ 1) posts left after diversification, that is, $r \cdot n$ non-redundant posts per λ_t time. We also assume that the each author generates the same number of $\frac{n}{m}$ posts with $\frac{r \cdot n}{m}$ left after diversification. Further we assume in the author similarity graph, each author has d neighbors and is in c ($\leq d$) cliques. We denote s as the average number of authors in a clique.

Note that cliques may have overlaps. If we define q as the number of edges in G over the total number of edges in c cliques from G, we have $\frac{m \cdot c}{s} = \frac{m \cdot d}{s \cdot (s-1) \cdot q}$, where both sides compute the number of distinct cliques. Thus we can expect $c \cdot (s-1) \cdot q = d$ with $0 \leq q \leq 1$.

Recall that UniBin puts posts from all authors in Z into a single post bin. Thus, the total bin size is $r \cdot n$ in UniBin. Each new post is sequentially compared to each post in the bin and thus the number of post comparisons per new post is $r \cdot n$. Each non-redundant post incurs one insertion into the bin.

NeighborBin maintains a set of per-author bins with each bin storing posts from an author and her similar authors. Roughly, each per-author bin stores $\frac{d+1}{m} \cdot r \cdot n$ posts. Thus the total number of post copies stored in memory is $(d+1) \cdot r \cdot n$. At the arrival of a new post P_i, the number of post comparisons made by NeighborBin is $\frac{d+1}{m} \cdot r \cdot n$ (compare P_i to all posts in $author(P_i)$'s post bin). Each non-redundant post incurs a total of $(d+1)$ insertions into the bins.

In CliqueBin, for each non-redundant post P_i we store its c copies: one copy in the bin of each clique containing $author(P_i)$. Thus, the total size of the clique bins is $c \cdot r \cdot n$. CliqueBin compares each new post P_i to posts in the bins of c cliques that contain $author(P_i)$, which leads to a total of $\frac{s \cdot c}{m} \cdot r \cdot n$ comparisons. Each non-redundant post incurs a total of c insertions into the bins.

Table 2 summarizes the performance analysis. We can see that all these results contain the same component $r \cdot n$. Obviously, all

Table 2: Performance estimation of the algorithms for SPSD

	UniBin	NeighborBin	CliqueBin
RAM	$r \cdot n$	$(d+1) \cdot r \cdot n$	$c \cdot r \cdot n$
Comparisons in λ_t	$r \cdot n^2$	$\frac{d+1}{m} \cdot r \cdot n^2$	$\frac{s \cdot c}{m} \cdot r \cdot n^2$
Insertions in λ_t	$r \cdot n$	$(d+1) \cdot r \cdot n$	$c \cdot r \cdot n$

three diversity thresholds effects the ratio of non-redundant post r. The value of n is affected by several factors, such as the frequency of the post stream P and the setting of time diversity threshold λ_t.

An important factor that affects the performance of the algorithms, especially NeighborBin and CliqueBin, is the topology of the author similarity graph G. In the above estimates, we use parameters d, c, s and m to capture the topology properties. We note that the values of the ratios of d, c, s to m are functions of the author diversity threshold λ_a. Given a set of subscribed authors (i.e., with m fixed), the larger λ_a the denser G is (in terms of the number of edges). Thus, the number of neighbors per author (d) increases with λ_a, which means the performance of NeighborBin will drop if all other settings remain unchanged. We also argue that c and $c \cdot s$ increase with the graph's density, and hence we expect CliqueBin to perform better for smaller λ_as. In Section 6, we confirm this through experiments on real data set.

In Section 6 we will summarize the use cases for each algorithm based on this theoretical analysis combined with our experimental results.

4.5 Summary

We summarize the characteristics of the three algorithms in Table 3. In terms of data structure, UniBin and NeighborBin need the author similarity graph, while CliqueBin needs the mapping of each author to the set of cliques containing the author. As we mentioned, we assume that all these data structures are maintained in memory.

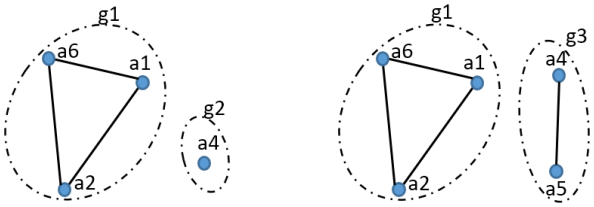
We can see that UniBin requires the least RAM. NeighborBin reduces the post comparisons compared to UniBin, but has high RAM consumption because it maintains multiple copies of a post. CliqueBin outperforms NieghborBin in terms of RAM consumption, by reducing the number of copies per post (and thus insertions per post), but it incurs more post comparisons. Since CliqueBin still maintains multiple copies of a post, it requires more insertions and higher RAM consumption than UniBin. Also, since CliqueBin does not compare posts from non-similar authors, we expect the number of comparisons in CliqueBin to be lower than in UniBin.

5. ALGORITHMS FOR MULTIPLE-USERS SPSD (M-SPSD)

In this section, we extend our ideas to solving M-SPSD. When we move from applying the diversity model for one user to multiple users, the crucial question is whether it is possible to reuse the computation performed for diversifying one user's stream to diversify the other users' streams.

A simple way to solve M-SPSD is to process the post stream for each user individually. That is, we can apply the algorithm for SPSD on each user's post stream separately. We denote the corresponding algorithms for M-SPSD as **M_UniBin**, **M_NeighborBin** and **M_CliqueBin** respectively, to distinguish them from the algorithms for SPSD. In this section, we present variations of these algorithms to optimize the diversification process by reusing computations for multiple users who share subscriptions.

If two users do not share any common subscriptions, then their post streams are disjoint and thus the computation of diversifying one's stream cannot be reused for diversifying the other users' post streams. Hence we only consider the cases for optimization when users share the same subset of subscriptions.



(a) G_1 (b) G_2

Figure 7: Author similarity graphs of two users u_1 and u_2.

However, we notice several limitations to reusing the diversification computation across multiple users, even if they share some subscriptions. We use examples to illustrate this. Figure 7 shows two users, u_1 and u_2, sharing a set of subscriptions {a1, a2, a4, a6}.

We notice that after diversification u_1 may see a different subset of the posts from a4 as u_2. u_2 subscribes to a5 which is a similar author to a4. Thus, it is possible that some posts from a4 are shown to u_1 but not to u_2 if they are covered by a5's posts.

However, the same diversified set of posts from {a1, a2, a6} will be shown to u_1 and u_2. The three authors form a *connected component* (denoted as g1 in Figure 7) in both G_1 and G_2. That is, in both G_1 and G_2 there are no other authors similar to any author in {a1, a2, a6}. Hence, posts from other subscribed authors can not cover the posts from {a1, a2, a6}. Thus, the diversification processes on the posts from {a1, a2, a6} are exactly the same for u_1 and u_2. This means that we can reuse the data structures and computation across u_1 and u_2 for diversifying the post stream from {a1, a2, a6}.

Based on these observations, we can optimize the diversification process for multiple users if they subscribe to a same set of authors that form a connected component. We can then consider a post stream (a subset of P) of each connected component separately, apply the diversification algorithm on it, and then merge the diversified post streams together.

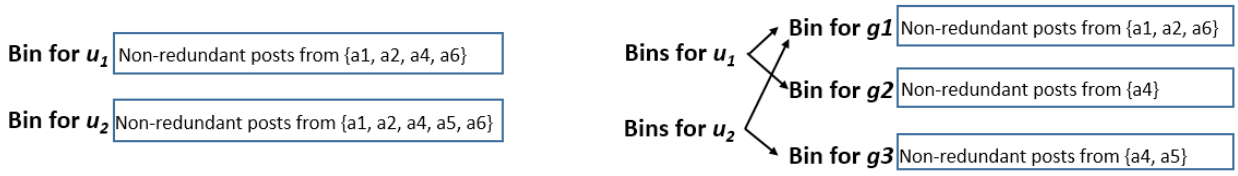
For this, we first process the author similarity graph G_i of each user u_i to compute all connected components of all G_is. (Since different G_is may overlap, some nodes may appear in several components.) For each distinct connected component g_i, we run one of the proposed algorithms for SPSD on the post stream by the authors in g_i. User u_i's post stream consists of the union of the diversified post streams from all connected components in G_i.

For example, as shown in Figure 8b, we can apply the UniBin algorithm for three distinct connected components (g1, g2 and g3), that is, we maintain a single post bin for each of the three components. Then the posts shown to u_1 is the union of the two diversified post streams from g1 and g2. We refer this algorithm as *S_UniBin*. For comparison, we show the example for M_UniBin in Figure 8a. M_UniBin maintains a post bin for each user seperately. To extend NeighborBin, we maintain a per-author post bin for each author in a distinct connected component g_i. To extend CliqueBin, we do the clique partition for each g_i, then maintain a per-clique post bin as described earlier.

We denote the three algorithms with the above optimization as **S_UniBin**, **S_NeighborBin** and **S_CliqueBin** respectively.

Table 3: Differences between the three algorithms for SPSD

		UniBin	NeighborBin	CliqueBin
Data Structures		(1) Author similarity graph (2) A single post bin storing posts from all authors.	(1) Author similarity graph (2) A post bin per author storing posts from the author and her neighbors.	(1) Author clique mapping (2) A post bin per clique storing posts from all the authors in the clique.
Properties	RAM	Low	High	Moderate
	Comparisons	High	Low	Moderate
	Insertions	Low	High	Moderate



(a) M_UniBin

(b) S_UniBin

Figure 8: Example of M_UniBin and S_UniBin.

6. EXPERIMENTAL EVALUATION

6.1 Data Set and Experimental Settings

We conducted our experiments on Twitter data. The authors in [22] published a Twitter social graph dataset consisting of more than 660,000 Twitter authors (accounts). Computing the author similarity graph for the whole data set would be prohibitive, as it requires comparing all pairs of authors. Instead, we used a subgraph of 20,150 authors obtained by randomly picking an initial author, and adding authors that are reachable through Breadth First Search on the follower-followee graph.

We computed all pairwise author similarity for these 20,150 Twitter authors. The author similarity distribution is depicted in Figure 9, where the x-axis shows the author similarity value and y-axis shows the fraction of author pairs with similarity values larger than the value indicated by x-axis. It shows that 2.3% author pairs are with similarity ≥ 0.2 and 0.6% pairs are with similarity ≥ 0.3.

Further, we crawled the tweets of these twitter authors using Twitter REST API[2] for one day. The tweets data set contains 233,311 tweets, which means these Twitter authors post slightly over 10 tweets per author per day. After we removed some short tweets that have less than two words or only contain meaningless tokens, there are 213,175 tweets left.

We implemented all algorithms in Java. We ran our experiments on machines with Quad Core Intel(R) Xeon(R) E3-1230 v2@3.30GHz CPU and 16GB RAM.

6.2 Performance of the algorithms for SPSD

In this section, we evaluate the performance of the three algorithms for SPSD. We assume that a user follows all the Twitter authors in our dataset, and we run the algorithms on the user's post stream which consists of 213,175 posts in one day.

First, we study the effect of the three diversity dimensions: time, content and author. Figure 10 shows the number of tweets left after diversification under different settings by removing diversity dimensions and varying diversity thresholds. Incorporating all three diversity dimensions with reasonable diversity thresholds, the diversification model prunes about 10% redundant posts. We notice that incorporating only some of these dimensions will largely change the size of diversified stream. It means that all three dimensions play an important role in diversifying tweet data.

[2]https://dev.twitter.com/overview/documentation

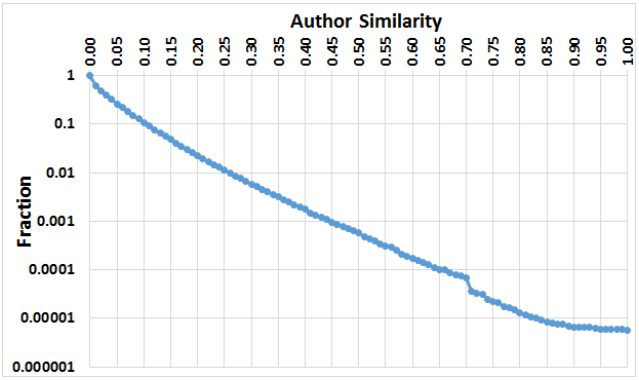


Figure 9: Author similarity distribution in our data set

6.2.1 *Performance of the algorithms under different diversity settings*

The analysis in Section 4.4 indicates that the performance of the three algorithms for SPSD is effected by several factors such as the diversity thresholds and the post stream throughput. These diversity settings could change the relative performance between the three algorithms. In this section, we study the performance of each algorithm under different settings and we experimentally show that each algorithm outperforms the other two in certain settings. Supported by former analysis and experimental results, we will summarize use cases for each algorithm.

Varying time diversity threshold λ_t. In Figure 11, we present the performance of UniBin, NeighborBin and CliqueBin under different time diversity thresholds (λ_t). In this experiment, we set $\lambda_c = 18$ (according to the results in Figure 4) and $\lambda_a = 0.7$ (i.e., we consider two authors are similar if the cosine similarity between

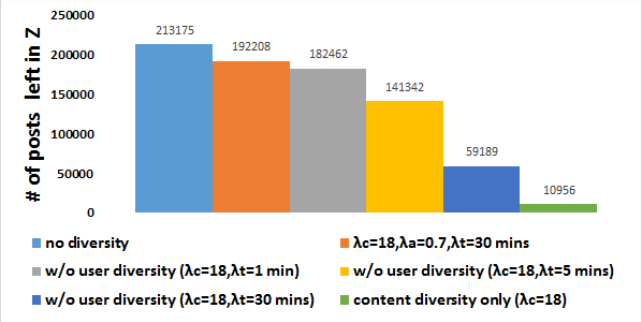
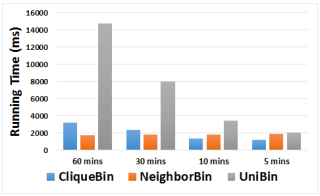


Figure 10: Number of tweets left after applying diversification in our data set

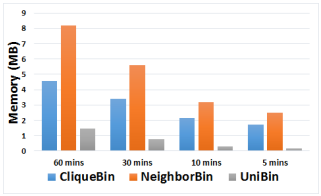
their followee vector is ≥ 0.3 and thus distance is ≤ 0.7). The running time shows the execution time for an algorithm to ingest the 213,175 posts.

In Figure 11a we can see that the running time of all three algorithms decreases with smaller λ_ts. The reason is that with a smaller λ_t, the algorithms perform fewer pairwise post comparisons (depicted in Figure 11c). NeighborBin and CliqueBin outperform UniBin in terms of running time. We also notice that CliqueBin is more efficient than NeighborBin when λ_t is small (e.g., $\leq$ 10 minutes). This gives us evidence for the summarization of use cases in Table 4 for NeighborBin and CliqueBin.

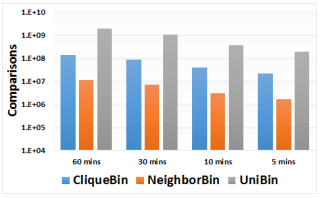
Smaller λ_t also reduces the RAM consumption because the algorithms store shorter history of Z in post bins. As expected, NeighborBin requires more memory than UniBin and CliqueBin.



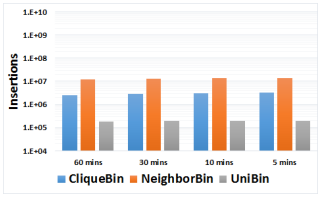
(a) Running time



(b) RAM



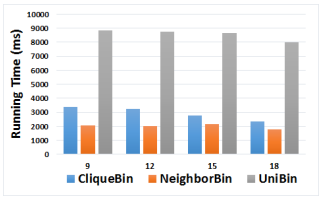
(c) Post comparisons



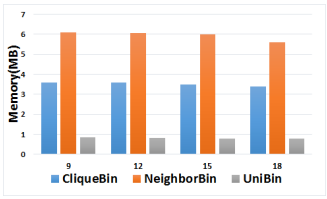
(d) Post insertions

Figure 11: Performance of the three algorithms under different time diversity thresholds λ_t.

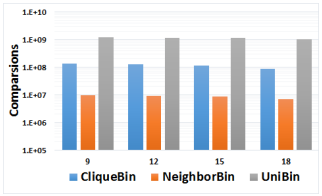
Varying content diversity threshold λ_c. We also study the performance of the three algorithms by varying λ_c. For this, we set $\lambda_t = 30\ mins$ and $\lambda_a = 0.7$ and we vary the λ_c from 9 to 18. Figure 12 depicts the results. It shows that, for all the three algorithms, the change of content diversity threshold only slightly affects the performance. The reason is that SimHash can effectively detect tweets with near-duplicate content for $\lambda_c \geq 9$ as we can see in Figure 4. With λ_c changing from 9 to 18, the precision is already stable. The recall is lower with smaller λ_c, which means more posts will be detected as non-redundant. But this increase in number of non-redundant posts is slight, and thus the increase in the number of comparisons and insertions does not affect the overall efficiency significantly.



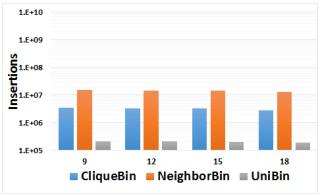
(a) Running time



(b) RAM

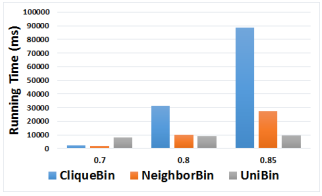


(c) Post comparisons

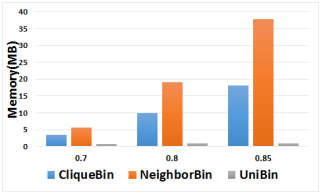


(d) Post insertions

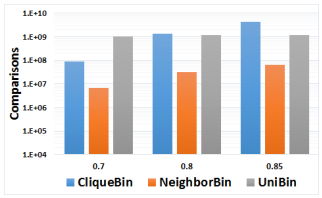
Figure 12: Performance of the three algorithms under different content diversity thresholds λ_c.



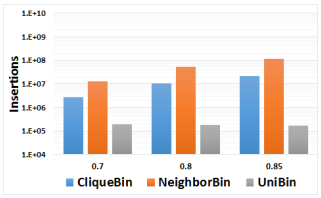
(a) Running time



(b) RAM



(c) Post comparisons

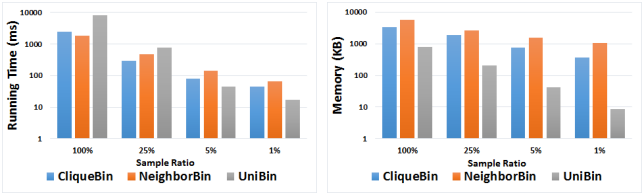


(d) Post insertions

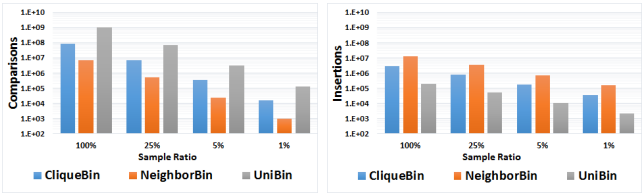
Figure 13: Performance of the three algorithms under different author diversity thresholds λ_a.

Varying author diversity threshold λ_a. Further, we study the performance by varying λ_a. The results are presented in Figure 13 where we set $\lambda_t = 30\ mins$ and $\lambda_c = 18$.

We observe that the author diversity threshold λ_a significantly affects the overall performance of NeighborBin and CliqueBin but not UniBin. When λ_a increases, the author similarity graph gets denser and thus the number of neighbors per author and the number of cliques per author both increase. For instance, when $\lambda_a = 0.7$ the number of neighbors per author (d) is 113.7, the number of cliques per author (c) is 29 and the average size of a clique (s) is 20 in our data set. They change to 437.3, 106 and 38 correspondingly with $\lambda_a = 0.8$. Hence, the number of copies per post in NeighborBin and CliqueBin increases. This explains that in Figure 13 the memory consumption by NeighborBin and CliqueBin

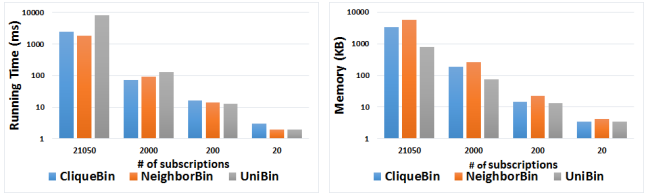


(a) Running time (b) RAM

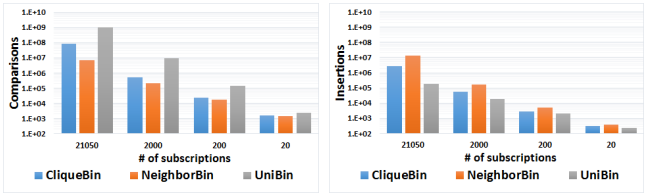


(c) Post comparisons (d) Post insertions

Figure 14: Performance of the three algorithms under different post rates.



(a) Running time (b) RAM



(c) Post comparisons (d) Post insertions

Figure 15: Performance of the three algorithms varying the number of subscribed authors.

increases sharply with larger λ_as. However, the number of non-duplicate posts does not vary much with different λ_as in our data set; thus the performance of UniBin is stable.

We note that when λ_a is large the performance of NeighborBin and CliqueBin (in terms of both memory consumption and running time) is significantly worse than UniBin. Hence, we expect UniBin is the best choice among these three algorithms in use cases where λ_a is large, as we summarize in Table 4.

Varying post stream throughputs. We also study the performance of the algorithms under different post stream throughputs. We test this in two ways: (i) varying subscriptions' post rate, and (ii) varying the number of subscriptions. For both, we keep $\lambda_t = 30\ mins$, $\lambda_a = 0.7$ and $\lambda_c = 18$.

Varying post generation rate. For this, we randomly sample the posts from the 21,050 authors and solve SPSD on the sampled post stream. We conduct experiments for the sample ratio 25%, 5% and 1% and present the results in Figure 14. The results show that when the throughput is low (the same ratio is low) UniBin outperforms the other two algorithms. We can also see that CliqueBin performs better than NeighborBin with a moderate or small post generation rate.

Varying the number of subscribed authors. The results shown above are for the case of one user subscribing (following) all Twitter authors in our dataset. In this experiment, we randomly sample Twitter authors in our dataset with different sample sizes. We assume that a user subscribes to all authors in one sample and we run the algorithms on the user's post stream. The results in Figure 15 show that UniBin slightly outperforms the other two when the number of subscriptions is small.

To summarize, UniBin delivers better performance than NeighborBin and CliqueBin when the stream throughput is low. This is consistent with our analysis in Section 4.4 – see also Table 4.

6.2.2 *Discussion*

Through extensive experiments, we observe that each algorithm outperforms the other two in certain cases. In Table 4 we summarize the best choice of algorithm in different use cases based on our analysis and experimental study.

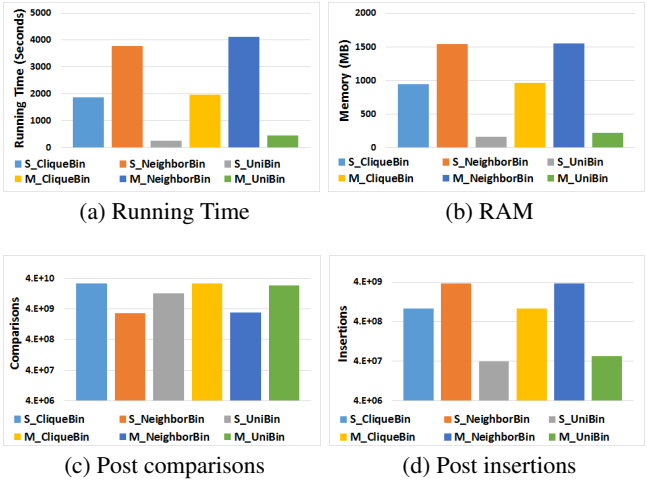
UniBin is the most memory efficient among the three algorithms. Thus in applications with limited RAM UniBin should be considered. Further, when the stream throughput is low (we tested it with small number of subscriptions and low post generation rate), UniBin performs better than the other two. According to the analysis in Table 2, we expect that the number of comparisons increases super-linearly with n (the number of posts in a λ_t time range), however the number of insertions increases sub-linearly with n. With a lower stream throughput (smaller n) the overhead of insertions in NeighborBin and CliqueBin is a large contribution to the total running time. When n is small enough, the overhead on insertions becomes larger than the saving on comparisons for NeighborBin and CliqueBin compared with UniBin. The similar reasoning can be applied to explain why UniBin is the best choice when λ_t is very small. To clarify, in Figure 11 we did not include the results by setting $\lambda_t = 1\ min$ where UniBin performs best among the three algorithms. We argued that with a larger λ_a both d (number of neighbors per author) and c (number of cliques per author) increase and thus NeighborBin and CliqueBin both have higher number of comparisons and insertions. Thus we can see UniBin is preferable when λ_a is set large. One example use case for UniBin is News RSS Feed reader, where the author similarity graph is dense. Generally, news agents form clusters (e.g., by their political views) such that in each cluster the news agents are similar to each other from a user's perspective. Another use case could be Google Scholar where the post (scientific publication) throughput is low.

In other cases, CliqueBin or NeighborBin will be the better choice. They both perform well in cases with a high or moderate stream throughput, which is very common for online social networks. The tie breaker between them is the time diversity threshold λ_t, as we analyzed λ_t determines the tradeoffs between costs of comparisons and insertions. CliqueBin is a better choice if λ_t is set moderately. For example, in Twitter information is time sensitive and thus people may be interested in reading posts with related content but with time distance larger than, say, minutes. For applications where the value of λ_t could be in hours or even days, NeighborBin can be applied. For example, Twitch[3] is a platform on which people can watch and share video game shows. Users may not be interested in watching the video record of the same match that posted at different time. Even in Twitter some users may prefer to customize the λ_t to a larger value, in order to reduce the post volume if they

[3]http://www.twitch.tv/

Table 4: Use cases of the three algorithms for SPSD

Conditions	*Algorithm choice*	*Example use case*
Very small λ_t OR low stream throughput OR large λ_a (dense G) OR RAM is a critical limitation	UniBin	News RSS Feed, Google Scholar
Large λ_t AND small λ_a (sparse G) AND high stream throughput	NeighborBin	Twitch
Moderate λ_t AND small λ_a (sparse G) AND high stream throughput	CliqueBin	Twitter



(a) Running Time (b) RAM

(c) Post comparisons (d) Post insertions

Figure 16: Performance of the algorithms for M-SPSD.

follow a large number of authors.

6.3 Performance of the algorithms for M-SPSD

We consider now the scenario where each Twitter author is also a user. Each user subscribes to (follows) a set of authors which we can get from the original follower-followee social graph. Then we run the algorithms solving M-SPSD for these 21,050 users in our data set. For the experiments in this section, we set $\lambda_t = 30\ mins$, $\lambda_a = 0.7$ and $\lambda_c = 18$.

The average number of subscriptions in our sampled user data set is 443.6 and the median is 187. Since we only crawled the posts and computed the author similarity graph for the set of 21,050 authors, we ignored the subscriptions that are not in this set. Then the average number of subscriptions per user drops to 130 and the median is 20. We should note that this reduces the probability of different users sharing common subscriptions.

Figure 16 presents the performance of the algorithms. It shows that the proposed optimization (reusing computation and data structure across multiple users described in Section 5) improves time efficiency as well as memory consumption. Specifically, S_UniBin uses 43% less running time and 27% less memory than M_UniBin. In the S_UniBin method, posts are stored separately by connected components. This reduces the number of comparisons significantly over M_UniBin. We also observe tthat S_NeighborBin reduces the running time of M_NeighborBin by 8% while S_CliqueBin improves M_NeighborBin by 4% in running time.

S_UniBin achieves superior performance. We also notice that S_NeighborBin requires fewer post comparisons than S_UniBin but many more insertions. We think that S_UniBin outperforms S_NeighborBin and S_CliqueBin also because its post access pattern is sequential while in the other two are not (each post bin is a map).

7. RELATED WORK

Time Aware Diversity. The authors of [7] solve the problem of maintaining the k most diverse results in a sliding window over a stream. MaxMin semantics is used. They maintain a data structure called the cover tree and show how to incrementally add new and remove expired results from this tree. The cover tree cannot be used for our diversity semantics because it cannot handle simultaneous similarity in three dimensions: time, content and author.

Diversification on Microblogging Posts. The work of [4] studies the problem of diversifying posts in microblogging systems. In their problem setting, users subscribe several queries (topics). However, in practice users are more often subscribing to authors, which is the setting of the problem we studied in this paper. In [4] they apply strict coverage semantics similar to ours, but limited only to time and content diversity. Unlike in our model, in [4] the content diversity is guided by the inputted queries where no inter-post content similarity is considered. They also studied the stream variation of their problem in which they allow a lag upon a new post to decide whether it should be outputted. In our problem, the diversity model is required to make the decision immediately at the arrival of a post.

Document Stream Summarization. The authors of [20] work on the problem of summarizing a Twitter stream. They model the summarization problem as a facility location problem. Give a budget of k, they aim to select k tweets that maximize the similarity to the whole tweets set. They incorporate the time factor to measure the document similarity of two posts. But unlike in our problem, instead of using a hard (boolean) threshold, they consider an exponential decay to the content similarity based on their timestamp difference. In the work of [13], the authors apply clustering techniques for Twitter stream summarization. Tweets are clustered according to content similarity. For each cluster, they build a word graph or phrase graph and pick frequent sentences ("paths" in the graph) to construct a summary. The sentences in the summary may not be in any original tweet. The authors of [18] propose a one-pass online clustering algorithm to cluster tweets, and then they generate online summaries by selecting k tweets (one from each cluster) that have high LexRank [8] score. In [16], the authors apply topic modeling for personalized time-aware tweet summarization. However, all these work do not consider author similarity to measure the similarity between tweets.

Detecting Duplicate Tweets. In [21], the authors propose to use machine learning methods to detect near-duplicates in tweets. For

this, they construct a rich set of syntactic, semantic and contextual features. They aim to distinguish different levels of near-duplicates, e.g. exact copy, strong near-duplicate, or weak near-duplicate.

8. CONCLUSION

In this paper, we studied the novel problem of diversifying social post streams by incorporating diversity in three dimensions: content, time and author. We illustrated the challenges of solving the problem and proposed various algorithms to efficiently handle these challenges. We showed the tradeoffs between our proposed algorithms and argued the use cases for them. We also studied the problem of applying the proposed diversification model for multiple users in a social system. Extensive experiments proved the effectiveness of our model and efficiency of proposed algorithms.

9. ACKNOWLEDGEMENT

Marek Chrobak is supported by National Science Foundation (NSF) grants CCF-1217314 and CCF-1536026. Vagelis Hristidis is supported by NSF grants IIS-1216007 and IIS-1447826.

10. REFERENCES

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 5–14, 2009.

[2] B. S. Alper, J. A. Hand, S. G. Elliott, S. Kinkade, M. J. Hauan, D. K. Onion, and B. M. Sklar. How much effort is needed to keep up with the literature relevant for primary care? *Journal of the Medical Library association*, 92(4):429, 2004.

[3] G. Capannini, F. M. Nardini, R. Perego, and F. Silvestri. Efficient diversification of web search results. *Proceedings of the VLDB Endowment*, 4(7):451–459, 2011.

[4] S. Cheng, A. Arvanitis, M. Chrobak, and V. Hristidis. Multi-query diversification in microblogging posts. In *17th International Conference on Extending Database Technology*, pages 133–144, 2014.

[5] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl. Divq: diversification for keyword search over structured databases. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 331–338. ACM, 2010.

[6] M. Drosou and E. Pitoura. Disc diversity: Result diversification based on dissimilarity and coverage. *Proceedings of the VLDB Endowment*, 6(1):13–24, Nov 2012.

[7] M. Drosou and E. Pitoura. Dynamic diversification of continuous data. In *15th International Conference on Extending Database Technology*, pages 216–227. ACM, 2012.

[8] G. Erkan and D. R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22(1):457–479, 2004.

[9] A. Goel, A. Sharma, D. Wang, and Z. Yin. Discovering similar users on twitter. In *11th Workshop on Mining and Learning with Graphs*, 2013.

[10] M. Koniaris, G. Giannopoulos, T. Sellis, and Y. Vasileiou. Diversifying microblog posts. In *Web Information Systems Engineering–WISE 2014*, pages 189–198. Springer, 2014.

[11] G. S. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*, pages 141–150. ACM, 2007.

[12] F. Morstatter, J. Pfeffer, H. Liu, and K. M. Carley. Is the sample good enough? comparing data from twitter's streaming api with twitter's firehose. In *Proceedings of the 7th International AAAI Conference on Web and Social Media*, 2013.

[13] A. Olariu. Clustering to improve microblog stream summarization. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on*, pages 220–226, 2012.

[14] M. G. Ozsoy, K. D. Onal, and I. S. Altingovde. Result diversification for tweet search. In *Web Information Systems Engineering–WISE 2014*, pages 78–89. Springer, 2014.

[15] D. Rafiei, K. Bharat, and A. Shukla. Diversifying web search results. In *Proceedings of the 19th international conference on World Wide Web*, pages 781–790, 2010.

[16] Z. Ren, S. Liang, E. Meij, and M. de Rijke. Personalized time-aware tweets summarization. In *Proceedings of the 36th international ACM SIGIR conference on research and development in information retrieval*, pages 513–522, 2013.

[17] C. Sadowski and G. Levin. Simhash: Hash-based similarity detection. Technical report, Technical report, Google, 2007.

[18] L. Shou, Z. Wang, K. Chen, and G. Chen. Sumblr: continuous summarization of evolving tweet streams. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 533–542. ACM, 2013.

[19] S. Sood and D. Loguinov. Probabilistic near-duplicate detection using simhash. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1117–1126. ACM, 2011.

[20] H. Takamura, H. Yokono, and M. Okumura. Summarizing a document stream. In *Advances in Information Retrieval*, pages 177–188. Springer, 2011.

[21] K. Tao, F. Abel, C. Hauff, G.-J. Houben, and U. Gadiraju. Groundhog day: near-duplicate detection on twitter. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1273–1284, 2013.

[22] X. Wang, H. Liu, P. Zhang, and B. Li. Identifying information spreaders in twitter follower networks. Technical Report TR-12-001, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ 85287, USA, 2012.

[23] X. Yang, A. Ghoting, Y. Ruan, and S. Parthasarathy. A framework for summarizing and analyzing twitter feeds. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 370–378. ACM, 2012.