Continuous processing of top-k queries in social networks

Abdulhafiz Alkhouli

Université de Cergy-Pontoise, Université de Paris Seine,
ENSEA, CNRS (UMR 8051)
ETIS, Equipes Traitement de l'Information et Systémes -
École doctorale Sciences et Ingénierie

THÈSE DE DOCTORAT

présentée pour obtenir le titre de DOCTEUR en Informatique

Abdulhafiz ALKHOULI

Continuous top-k queries in social networks

Directeur de thèse
Dan VODISLAV, Professeur, laboratoire ETIS

Co-encadrant
Boris BORZIC, Ingénieur de recherche, laboratoire ETIS

Date de soutenance : 29 Septembre 2017

COMPOSITION DU JURY

M.	Bernd AMANN, Professeur, laboratoire LIP6, Université Pierre et Marie Curie, Paris	Rapporteur
M.	Jean-Marc PETIT, Professeur, laboratoire LIRIS, INSA Lyon	Rapporteur
Mme.	Salima BENBERNOU, Professeure, laboratoire LIPADE, Université Paris Descartes	Examinatrice
M.	Cédric DU MOUZA, MCF HDR, laboratoire CEDRIC, CNAM Paris	Examinateur
M.	Dimitris KOTZINOS, Professeur, laboratoire ETIS	Examinateur
M.	Dan VODISLAV, Professeur, laboratoire ETIS	Directeur de thèse
M.	Boris BORZIC, Ingénieur de recherche, laboratoire ETIS	Co-encadrant

Contents

Abstract

Information streams provide today a prevalent way of publishing and consuming content on the Web, especially due to the great success of social networks. In the social networks context, users may subscribe to several information sources of interest and continuously receive new published content. But, this new publishing/consumption mode may lead to huge amounts of received information, overwhelming for human processing. Thus, there is a vital need to develop effective filtering and ranking techniques which allow users to efficiently be updated with the most interesting content.

Continuous top-k queries over the streams of interest allow limiting results to the most relevant content. To provide a relevant content, the ranking model should consider various context factors including the traditional content-based ones, but also time-based and social network factors. In the social network, maintaining top-k sets may be more difficult because of their dynamic context: users not only publish new messages, but may also interact with existing ones, modify relations, change the profile, etc. For a large social network with millions of users and billions of messages, the continuous processing of the continuous top-k queries is the most effective approach. However, current systems fail in combining continuous top-k processing with rich scoring models including social network criteria. Moreover, such systems do not consider the diversity of published content.

In this thesis, we focus on filtering information streams based on the computation of top-k messages for each user in the social network. We aim at developing a scalable system able to efficiently evaluate the continuous top-k queries with a complex scoring function. We propose the SANTA algorithm, able to handle scoring functions including content similarity, recency, but also social network criteria and events in a continuous processing of top-k queries. We propose a variant (SANTA+) that accelerates the processing of interaction events in social networks. To provide both diverse and relevant messages in top-k sets, we propose the DA-SANTA algorithm which extends the SANTA algorithm to integrate the diversity into the continuous top-k model while maintaining the efficiency of the system. Our experiments are conducted over a real dataset extracted from Twitter, illustrating the properties of our algorithms and demonstrating their efficiency.

Résumé

En raison du grand succès des réseaux sociaux, la nature et mode de diffusion de l'information sur le Web a changé en faveur de contenus dynamiques diffusés sous forme de flux d'information. Dans le contexte des réseaux sociaux, les utilisateurs peuvent s'abonner à de multiples sources d'information et recevoir continuellement de nouveaux contenus. Or, ce nouveau mode de publication/consommation peut entraîner d'énormes quantités d'information, en surchargeant les utilisateurs. Ainsi, il est essentiel de développer des techniques efficaces de filtrage et de classement qui permettent aux utilisateurs d'être efficacement mis à jour avec le contenu le plus intéressant.

Les requêtes top-k continue sur les flux d'information limitent les résultats au contenu le plus pertinent. Pour améliorer la pertinence des résultats, le modèle de classement des résultats de requêtes devrait tenir compte de divers facteurs de contexte, y compris les facteurs traditionnels basés sur le contenu, mais aussi sur le temps les facteurs liés aux utilisateurs et à leurs relations (réseau social). Dans le réseau social, le maintien des ensembles de top-k peut être plus difficile à cause de son caractère dynamique: au-delà de la publication de messages, on peut interagir avec des messages existant, modifier les liens avec d'autres utilisateurs, changer son profil, etc. Pour un grand réseau social avec des millions d'utilisateurs et des milliards de messages, le traitement continu des requêtes top-k continue est l'approche la plus efficace. Cependant, les systèmes actuels pour le traitement continu des requêtes top-k ne considèrent pas des modèles de classement riches, incluant des critères de réseau social. En outre, de tels systèmes ne tiennent pas compte de la diversité des contenus publiés.

Dans cette thèse, nous nous concentrons sur le filtrage des flux d'information basé sur le calcul des messages top-k pour chaque utilisateur dans le réseau social. Nous visons à développer un système à large échelle capable d'évaluer efficacement les requêtes top-k continues avec une fonction de classement complexe. Nous proposons l'algorithme SANTA, capable d'utiliser des fonctions de classement, incluant la similarité du contenu, le temps mais aussi les critères et les événements du réseau social avec un traitement continu des requêtes top-k. Nous proposons aussi une variante (SANTA +) qui accélère le traitement des interactions avec les messages dans les réseaux sociaux. Pour fournir des réponses pertinentes et diverses, nous proposons l'algorithme DA-SANTA qui étend l'algorithme SANTA pour intégrer la diversité dans le modèle top-k continu tout en maintenant l'efficacité du système. Nos expérimentation sont menées sur des données réelles extraites de Twitter, permettant

d'étudier les propriétés de nos algorithmes et de montrer leur efficacité.

1 Introduction

1.1 Information streams

Web 2.0 technologies have deeply changed the way information is published and consumed on the Web. Content publishing takes more and more the form of information streams available through various information channels. As a result, information consumption becomes per stream where users receive in real-time contents from different information sources. Information streams consist of flows of items, usually short semi-structured text messages, possibly containing links to some Web resources (images, videos, pages, etc.), and continuously published through specific diffusion channels, e.g. RSS feeds from media, blogs, discussion forums, social networks, etc. Users may subscribe to several information channels of interest and continuously receive new published content. Users may also query this dynamic streams to retrieve relevant information. Every moment, billions of messages being published with a variety of topics and millions of users expect to receive in real-time a stream of the most interesting messages. For example, Twitter[1] has 320 million active users with 340,000 tweets per minute[2].

This dynamic content generation and consumption has continuously gained importance compared to traditional Web publishing (of Web pages) and exploring (through bookmarks, search engines and hyperlink navigation). Nevertheless, the new publishing/consumption mode may lead to huge amounts of received items, overwhelming for human processing. Thus, there is a vital need to develop effective filtering and ranking techniques which allow users to efficiently be updated with the most interesting items. Effective processing of the enormous amount of messages in information streams has generated considerable recent research interest [CNN+10, CC15, BOPY07, HMA10]. One of the main objectives of these studies is how to provide an interesting content for a large number of users and ensure effective filtering techniques given this unprecedented amount of messages.

Nowadays, users play a key role in the production and consumption of information streams. Previous passive readers have become both active information collectors and producers. With the great success of the social network, users actively participate in providing information. In social network services, users can have several social activities such as publishing a message in Twitter, uploading a video on

[1] www.twitter.com

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

Youtube[3], tagging a picture in Flickr[4] or sharing a message in Facebook[5]. This user-generated content contributes to the exponential growth of the Web as seen in Figure 1.1 which shows the large amount of data generated by several social services every minute in 2016. In this context, the large number of social network users becomes a challenge in the ranking techniques. The collection of the generated content in the social network, ranking, and filtering of such large-scale multi-query environment should be made in real-time. In addition, user queries could be implicit based on user profiles that consider long queries, which have a significant impact on the effectiveness of the filtering approach compared to small queries.

A common behavior in almost all social networks is that each user follows other users "friends" in order to be informed of newly published content. Consequently, users receive real-time content published by their friends. An important dimension in this publish-subscribe (pub/sub) framework is the relationship between publishers of information streams and subscribers. This social network dimension varies from no relationship at all in the case of RSS/Atom feeds, to possible interaction with the published messages on blogs (comments) and discussion forums (reply messages), and to explicit relationships between users playing the double role of publishers and subscribers on social networks such as Facebook (symmetric "friendship" relations) or Twitter (asymmetric "following" relations). The social network dimension contributes not only with providing information streams of interest to end users, but also comes with criteria to measure this potential interest.

Furthermore, social networks allow users to evaluate content by giving feedback at different levels. Users can produce different types of feedback on the published content, e.g. they can like a message, comment on it or share it. Each of these types plays an important role in assessing the importance of the published content and the visibility of information. Many research works in the web search domain have focused on integrating the social network dimension and social feedback in the filtering techniques in order to personalize the results and to improve their quality [YLL10, GCK+10, KS12]. Unfortunately, these studies fail to process in real-time a huge amount of information for a large number of users.

One way to cope with the huge amount of information, potentially interesting, available from different information channels is to organize the channels of interest by topic. Mechanisms such as RSS/Atom allow users to subscribe to information sources; once a new information is available in the subscriptions sources, users will immediately be notified. By using RSS reader applications such as Feedly[6], users can group the information sources by topic (e.g Politics, Science, Tech, etc.), to help users to choose information streams of interest. This is impractical since it limits the number of the subscriptions sources made by users, otherwise users would still be overloaded by the published content. Also, in each channel not all the published

[3]www.youtube.com
[4]www.flickr.com
[5]www.facebook.com
[6]www.feedly.com

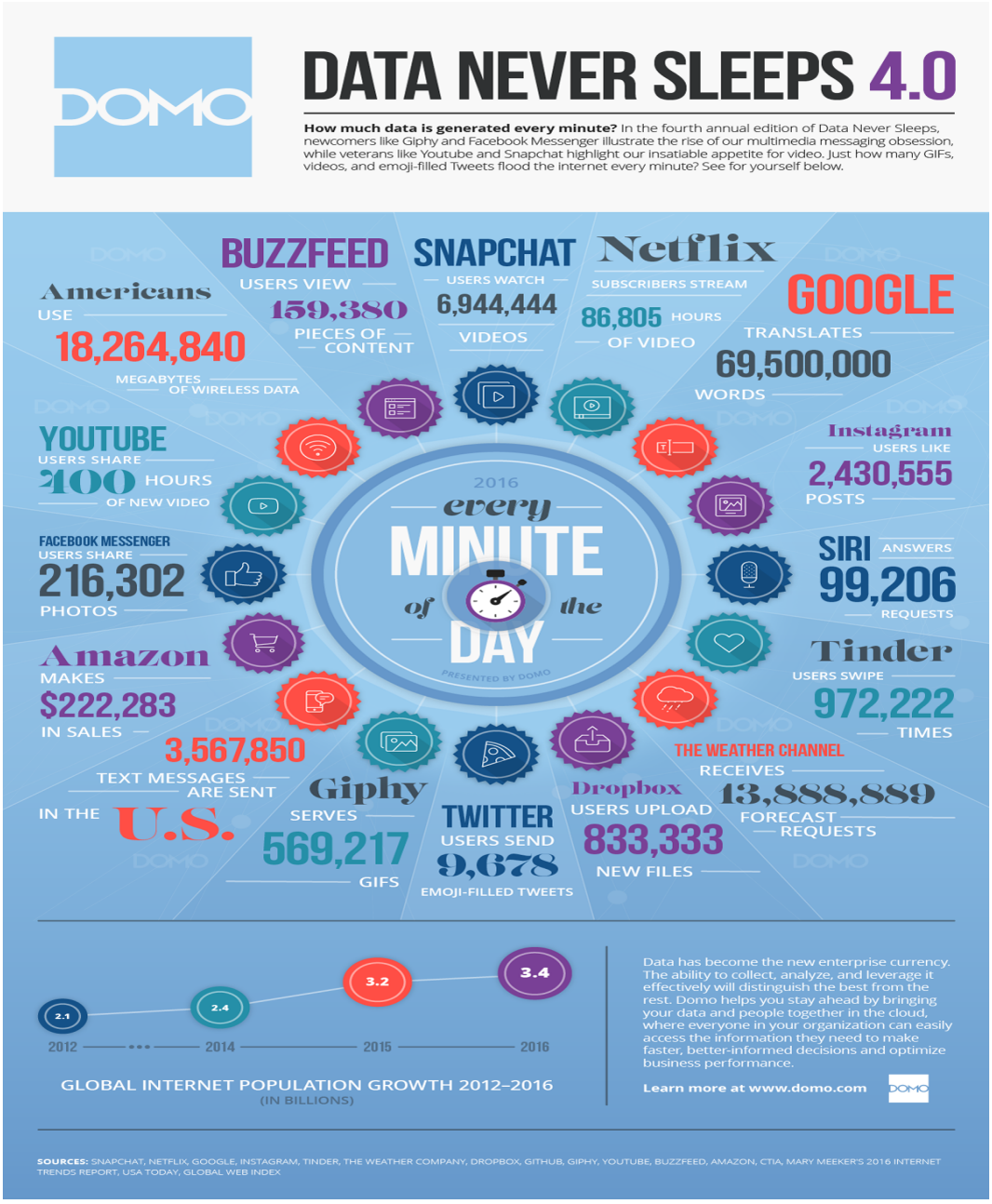


Figure 1.1: Content generated by users every minute in 2016 source: https://www.domo.com/learn/data-never-sleeps-4-0

content is equally useful to users. Thus, models for ranking content and filtering are necessary.

Recent research studies propose methods for filtering information streams. Boolean filtering *model* has been first proposed for text streams, using filters based on boolean keyword predicates. However, it is difficult to manage the number of results because in one context it may retrieve too many results and too few in another context. An alternative way to organize the large amounts of stream messages is to define a ranking model based on the importance of a message relative to a given subscription query. Measuring this importance by a relevance score allows end users to identify and to focus on the most important messages for them. Some approaches [ZC01, YGM99] limit the number of query results through a *predefined threshold* for the relevance score, only the messages that pass the *threshold* are presented to users. Nevertheless, choosing a threshold value to be suitable for a given context is a very hard task.

More recent works over information streams [MP11, RCCT14] consider only the k most relevant results in (pub-sub) publish-subscribe framework. This thesis adopts the filtering of the stream messages based on the *continuous top-k model. This model continuously maintains the k most relevant messages of the stream for each subscription query in the pub-sub system.* In this context, the first significant challenge is to define the ranking model since it plays a key role in determining the complexity and quality of results in the system.

The ranking model may depend on various context factors, among which we emphasize the following ones:

- *Content* based factors, measuring the adequacy of the message content with the subscription query. Since textual content is characteristic to information streams, content-based subscription queries are usually based on sets of terms of interest, and the importance of a message is evaluated from an information retrieval perspective, as the relevance of the text message to the query, based on popular models such as tf-idf [SB88] or BM25 [JWR00].
- *User* based factors, measuring the importance of users and of their relationships in the social network, for instance the importance of the message publisher and of the relationship between the subscriber and the publisher. In most cases user-based importance is measured on the social network graph, by evaluating e.g. node centrality and distance between nodes.
- *Interaction* based factors, measuring the importance of messages by the reaction they provoked, expressed through actions of other users on that message. Depending on the social network context, current actions may be likes, comments, forwards, tagging as favorite, etc.
- *Time* based factors, measuring the decrease of importance for a message as time goes by. Two main approaches are used to take into account this dimension: sliding time windows [GÖ03], resulting in dropping messages older than

a given duration, and time decay functions [SGFJ13, VAC12], expressing a continuous decrease of importance.

Other context factors, that we do not consider here, may contribute to evaluate the importance of messages, such as geographic location or other information elements specific to the social network and to the pub/sub environment. In Chapter 3, we will describe in detail our ranking model and how to implement these different factors. We will also see how to manage in particular the last two factors since these factors are dynamic (change over time).

1.2 Processing models

The second main challenge in the *continuous top-k model* for filtering information streams is the design and implementation of efficient processing models at a very large scale (millions, up to billions of users and information streams). In the case of ranking models based on scoring functions, where subscription results are limited to the most important messages, the main difficulty comes from the need of continuously (re-)computing the score of every message relative to every subscription query and of subsequently maintaining the lists of subscription results. The complexity of this task depends not only on the number of messages and queries, but also on the form of the scoring function.

Two main approaches of processing models have been proposed in the literature and in the commercial systems. The *static* approach runs periodic snapshot queries on all published messages to get the list of the most important messages for each user. The *continuous* approach handles subscriptions as continuous queries reacting to new messages and to other events, in order to incrementally maintain the important messages. If the continuous approach is more efficient, it also has more difficulties to handle complex scoring functions. To the best of our knowledge, the continuous methods proposed so far only explored simple scoring functions, most of the time based on the textual content, eventually combined with time factors. More complex scoring, including social network factors has been proposed, but only handled through a static approach.

Figure 1.2 presents the general architecture of continuous top-k processing of information streams, behaving as an event-based system. The result of such a process is the set of top-k messages for each user in the social network, continuously maintained by the system. The event processor handles every input event that may produce changes to the result lists, computes changes and subsequently updates the result lists. Change computation is based on the data structures representing the information streams and the social network, and on the index structures that enable efficient event processing.

In our information stream context, we distinguish two categories of events:

- *Continuously handled events*, with potentially strong impact on top-k update, and that must be processed on the spot. We include in this category, the publication of a new message and the interaction with an existing message.
- *Secondary events*, with a weaker impact on the top-k lists; they may be accumulated and processed from time to time. We include in this category changes in the social network that may produce small changes in the scoring parameters.

Evaluating the impact of various categories of events depending on the scoring model is a difficult problem, but continuously reacting to any event that may change some message score component is not realistic in practice, given the complexity of our scoring function. The above classification of events is a necessary trade-off between efficiency and precision.

In Chapter 3, we present our model for continuous top-k processing and we describe the SANTA algorithm, able to handle scoring functions taking into account the above factors. We also present a variant (SANTA+) that improves the processing of interaction events in social networks.

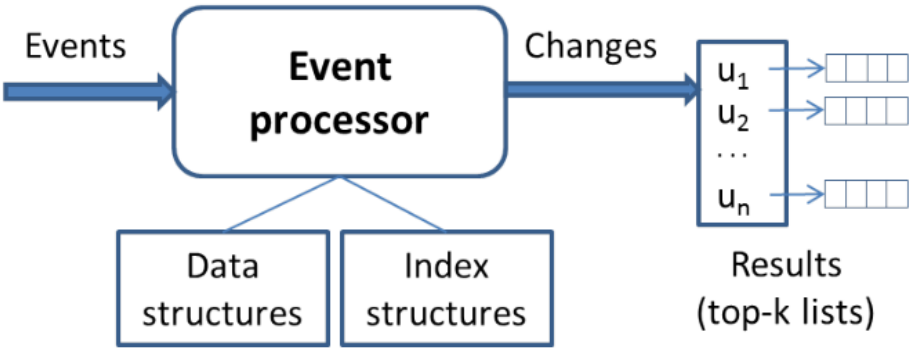


Figure 1.2: General architecture for continuous top-k processing of information streams

1.3 Diversification

Despite the positive effect of the proposed scoring models to improve the filtering process and increase user satisfaction in the top-k pub/sub system, such models do not guarantee that messages in top-k are diverse. Indeed, ranking based solely on relevance can often produce results that are too homogeneous, which could lead to the over-specialization problem. Moreover, in the social network environment, when an important event occurs, for example "Brexit", stream messages published on this event tend to be similar due to the nature of the actions in this environment, i.e sharing, retwetting, tagging, liking, etc. Thus, users may receive similar and redundant messages in real time.

The diversification of results is widely studied in traditional IR system and web search to improve the results quality [VRB+11, AK11, GS09]. The aim of diversification is to ensure that the query results are the most relevant and, at the same time, as diverse as possible. In the literature, several forms of the diversity have been studied. *Content-based* diversification focuses on increasing dis-similarity between documents in search result. *Intent-based* diversification interprets explicitly all possible intents (topics) of user's query using external information such as query logs, and retrieves documents that satisfy all these intents. In this thesis, we focus on the content-based diversification in order to avoid duplicate content in the top-k lists.

In general, a new message can enter a top-k of a given user, if it has a good enough relevance score. In *content-based* diversification, a new criteria (diversity) is added to the relevance to rank the messages in the top-k list. The relevance expresses the importance of a message for a user query, diversity could express the distance of a message to other messages in top-k. This creates a new filter for stream messages because we want to avoid similar messages in the results, even if the new message has a good relevance score to be ranked high in the top-k, it can be excluded if its diversity is low, i.e it is very similar to existing messages in top-k. In order to combine these two criteria, we adapt bi-criteria objective functions ([GS09]) that mix the relevance and diversity in a linear way.

In information streams context, few researchers have addressed the problem of diversity for a large number of queries. To our knowledge, all of top-k pub/sub systems excepting [CC15], do not consider the diversification of results. In the second part of the thesis, we focus on the diversity alongside the relevance in the computation of top-k. We develop an effective solution to scaling up on the number of stream messages and users. In Chapter 4, we present the DA-SANTA (Diversity-aware SANTA) algorithm, which is able to continuously maintain the k messages that are both diverse and relevant for each subscription query in the top-k pub/sub system.

1.4 Thesis focus

In this thesis, we focus on filtering information streams in a social network context based on the computation of top-k messages for a large number of users. We follow the *continuous* approach to evaluate the top-k queries using a rich scoring function mixing content-based, user-based, interaction-based and time-based components. We also tackle the diversity problem to improve the effectiveness of results and increase the user satisfaction. We model the social network as a pub/sub environment, where users publish messages and subscribe to information streams produced by other users in the network. The subscription queries are implicit, based on the user profile. In this model, when a new message is published by a user, all other users are candidates. We aim to achieve two main objectives. First, to develop a scalable method to maintain the k most relevant messages for each user in the social

network at all time. Second, to introduce diversity into this model in order to ensure that messages in top-k lists are relevant and diverse while preserving the efficiency and the quality of the results.

The main contributions of this thesis are the following ones:

- We propose a general model of *information stream social network (ISSN)* covering many existing social networks. In an ISSN model, stream messages are generated by social network users and relations between users are non-symmetric. User profiles constitute implicit queries over message streams. In order to filter/rank stream messages, we propose a rich scoring model including all the factors introduced above: textual content importance, user relative importance, interaction importance and time factor.

- Based on this model, we propose an algorithm, SANTA (Social and Action Network Threshold Algorithm), for continuous top-k processing of information streams, using a scoring function that includes all the social network factors identified above, and able to handle both publication and action events - to the best of our knowledge, SANTA is the first algorithm of this type. SANTA adopts a simple sorted-lists index that provides flexibility for the extension to new score criteria in a social network context, and for parallel processing. To enable efficient processing for a new event, we use an index structure for queries (user profiles). SANTA using threshold-based techniques evaluates the new event with a limited number of queries in the index. Existing algorithms for continuous top-k processing [VAC12, HMA10, SGFJ13, MP11] can be hardly extended with social network criteria in the scoring function and face the problem of heavy index updates when the top-k scores change. Unlike them, SANTA minimizes index updates by isolating changes into a single dimension.

- A variant of SANTA, called SANTA+, that significantly improves the processing of action events. SANTA+ avoids re-processing messages by maintaining an additional data structure to store the processed messages with information calculated when the message is published.

- The DA-SANTA (Diversity-aware SANTA) algorithm extending the SANTA algorithm for top$-k$ continuous processing, and is able to continuously maintain the k messages that are both diverse and relevant for each query in the system. The DA-SANTA algorithm adopts the commonly used *max-sum diversification* bi-criteria objective function [GS09] to combine relevance and diversity into a single scoring function for top$-k$ computation.

- A rich set of experiments over a real dataset extracted from Twitter, illustrating the properties of our algorithms and demonstrating their efficiency compared to an extension of a state-of-the-art algorithm.

1.5 Thesis outline

Chapter 2 presents a literature survey of the related work. The first part of this chapter presents a review in the context of the continuous top-k queries for information streams. We present the filtering and ranking models proposed over text streams. For the top-k queries, we first introduce the general algorithms for processing top-k queries. We also present new filtering approaches used when the streams are produced in social networks. Then, we introduce the ranking models based on social network criteria in the search on the Web. We present the works that addressed the problem of processing top-k queries in the context of the social network. Finally, we present the closest studies to this thesis, the continuous top-k queries over information streams. The second part presents the problem of query results diversity. We first present a general introduction to the problem. Then, we introduce approaches for computing the diverse top-k result over a static collection of data. Finally, we present the continuous version of the query results diversity.

Chapter 3 describes our first contribution. We present our model of information stream social network (ISSN), the ranking model including social network criteria, and the SANTA/SANTA+ algorithms. Finally, we compare our algorithms with an extension of the closest approach in the state of the art, the COL-Filter algorithm.

Chapter 4 describes the second contribution. We present the problem of query results diversity, in particular, the continuous version of the diversification. We introduce the adopted diversity model. Then, we describe the DA-SANTA algorithm. Finally, we present the experimental evaluation of our approach.

Chapter 5 concludes and provides a discussion about possible future works.

2 State of the art

This work aims at realizing two main objectives. The first one is to propose an efficient solution for filtering content over text streams based on top-k selection with an efficient continuous processing approach. The proposed ranking function in the top-k model considers social network factors combined with text relevance factors. The second objective is to consider the results diversification problem in this continuous top-k model, while preserving the efficiency and effectiveness of our model.

This Chapter is divided into two parts. In the first part, section 2.1 presents an overview of the continuous top-k queries for information streams. In the second part, section 2.2 presents the problem of query results diversity.

2.1 Continuous top-k queries for information streams

Publishing and consuming content through information streams is today at the heart of the new Web. Information streams consist of flows of items, usually short semi-structured text messages, possibly containing links to some Web resources (images, videos, pages, etc.), and continuously published through specific diffusion channels, e.g. RSS feeds from media, blogs, discussion forums, social networks, etc. Users may be both producers and consumers; as consumers, they subscribe to information channels of interest and continuously receive on it, in real-time, new published content. Nevertheless, this new dynamic publishing/consumption mode may lead to huge amounts of received items, overwhelming for human processing. Thus, there is a vital need to develop effective filtering and ranking techniques which allow users to efficiently be updated with the most interesting items.

Our first goal is to propose an efficient solution for filtering content over text streams based on top-k selection with an efficient continuous processing approach. We distinguish in general two types of top-k queries, *snapshot top-k queries* and *continuous top-k* queries.

- *Snapshot top-k queries* return the k most relevant results from a static collection of items at a moment in time. The top-k results do not change in time.
- Continuous top-k queries are applied over a stream of items. These queries maintain the k most relevant results of the stream at all time. That is, continuous queries continue to update the top-k results when new items are available.

Many of algorithms are proposed for processing *snapshot top-k queries* in databases and information retrieval systems. In the case of multicriteria ranking, the most popular algorithms are Threshold Algorithm (TA) and No Random Access (NRA) [Fag02], presented below.

In this thesis, we are interested in the *continuous top-k queries* over information streams in the social networks. As mentioned above, two main categories of *processing models for the continuous top-k queries* have been proposed to date, the *static approach* and *continuous approach*.

1. The *static approach* is based on periodic snapshot queries over the set of published messages to get the top-k list for each query. In this model, the published messages are maintained in some index structure. Then, with a predefined scheduling strategy, a top-k algorithm is applied on the indexed messages for computing the top-k set from scratch for every subscription query. One of the major challenges is that messages arrive in real time and with a very high throughput. Maintaining all messages in an index structure is very costly. Also, the periodic evaluation of a large number of queries from scratch is clearly inefficient. The *static approach* is used in Twitter's search service, see subsection 2.1.6 for more details.
2. The *continuous approach* handles subscriptions as continuous queries reacting to new messages and to other events, in order to incrementally maintain the important messages. Here, only the subscription queries are indexed, and the incoming events are processed on-the-fly. Only the top-k results for the impacted queries are incrementally updated when a new event arrives to the system.

Recent works over information streams [MP11, HMA10, RCCT14, VAC12, SGFJ13] have adopted the continuous processing approach for top-k computation. If the *continuous approach* is more efficient, it also has more difficulties to handle complex scoring functions. The continuous methods proposed so far only explored simple scoring functions, most of the time based on the textual content, possibly combined with time factors. More complex scoring, including social network factors has been proposed, but only handled through the static approach.

We organize this section as follows: Subsection 2.1.1 presents filtering techniques used over text streams, in particular, the boolean filtering model and the filtering based on ranking models (*top-k filtering*). In Subsection 2.1.2, we present the popular algorithms for processing *snapshot top-k queries* in databases and information retrieval systems. In Subsection 2.1.3, we present specific filtering techniques and recommender systems when text streams are produced in a social network. Subsection 2.1.4 presents works that focused on the personalization of web search results and ranking models based on social network criteria. Subsection 2.1.5 introduces techniques and approaches for processing *snapshot top-k query* that use ranking models based on social networking criteria. In Subsection 2.1.6, we introduce top-k processing for *continuous top-k multi-query* environment based on the *static ap-*

proach. Subsection 2.1.7 presents continuous processing of top-k queries for a large scale pub/sub environment.

2.1.1 Filtering text streams

Several approaches have been proposed to tackle the problem of reducing the amount of information received from different streams by filtering their contents. We present in this section the most important filtering models proposed over text streams.

The *Boolean filtering model* has been first proposed for text streams, using filters based on Boolean predicates. In this model, the query is expressed by boolean expressions on terms, the terms are equally weighted. All items that match the query predicates, will be retrieved without any relevance ranking. In a pub/sub context such solutions [HKC+12, YGM94b] come with various index structures based on inverted lists for fast detection of the subscription queries concerned by a stream input message and avoid checking all subscriptions queries, in the context of a large number of subscriptions. The drawback of boolean filtering is that the number of results may be in some cases too big or too small.

Filtering based on ranking models. By using ranking models based on relevance scores, it is possible to select the best results and to adapt their number to the needs of the end users. Information retrieval (IR) ranking models, such as tf-idf [SB88] or Okapi BM25 [JWR00] provide ranking of results through a relevance score computed for each item in the context of a given text query. Usually, the Vector Space Model is used to represent queries and items and the cosine similarity function is considered to compute the query-item text relevance score [BM96].

Two main approaches have been proposed for filtering stream messages in an IR ranking context. The first one uses a predefined threshold for the relevance score [YGM94a, YGM99, Cal96, SB88] and it only keeps items in the query's result with a query-item similarity score that exceeds the threshold. However, finding the right threshold in a given context is a difficult task. A low threshold value may make the user be overloaded with an important amount of results, on the other hand, a high threshold value return no results at all. For this reason, [ZC01] proposes a method for adaptive detection of this threshold.

The second approach is a top-k computation that keeps only the k most relevant items in the query's results. The difficulty in this case, compared to threshold-based ranking, is to continuously maintain a changing list of top-k results.

In this context, our work addresses top-k filtering for information streams in a social network environment, going beyond text-only scoring models. We propose a social relevance score to compute the relevance of a message in the social network environment. Our purpose is to go beyond the state of the art methods for continuous processing of top-k queries over information streams, by considering a social

network environment with complex scoring functions that include the ranking factors mentioned above: content-based, social network based criteria and time-based factor.

2.1.2 Top-k multicriteria algorithms

In this section, we introduce the most popular algorithms for processing *snapshot top-k query* based on multiple criteria used in databases and information retrieval systems. The most known algorithm in this context is the Threshold Algorithm (TA) [Fag02]. Our work and many of the top-k algorithms for information streams that will be presented below, are based on the TA algorithm. In the following, we explain the TA algorithm from the perspective of information retrieval.

Given a finite set of documents D, for a document $d \in D$ having m terms $Td = \{t_1, ..., t_m\}$, we note $w_{t_1 d}, ..., w_{t_n d}$ the weights of terms in d (each term weight in the interval $[0, 1]$). Given a query q with n terms $Tq = \left\{t_1^{'}, ..., t_n^{'}\right\}$, with $w_{t_1^{'} q}, ..., w_{t_n^{'} q}$ the weights of terms in q. The overall relevance score for a document d relative to q is $f(w_{t_1^{'} q}, ..., w_{t_n^{'} q}; w_{t_1 d}, ..., w_{t_n d})$. Given a fixed number k, the top$-k$ algorithm should find the best k documents with the highest overall score.

In information retrieval, the TA algorithm supposes the documents are indexed in an *inverted index* sorted by the term weights. Let ω be the number of all terms appearing in D, the *inverted index* consists of ω sorted lists $l_1, ..., l_\omega$. Each term t maps to a list l that indexes documents containing t, sorted in the descending order of the term weight where each entry in the list has the form (d_i, w_{td_i}). The TA algorithm requires the aggregation function f to be monotonic, which is the case with cosine similarity or with BM25.

At query evaluation, the algorithm identifies the lists corresponding to the query terms. The TA algorithm traverses the lists as follows: it sequentially accesses the first entry of each of the lists, then the second entry of each of the lists, and so on. For each document seen in the index, the overall score is computed and if this score is greater than the $k - th$ score in the current top-k then the document can be added to the top$-k$ list. To avoid evaluating all the documents in the lists, TA examines a stopping condition at each step. Let $(\overline{d_i}, \overline{w_{t_j}})$ be the last entry seen in each list l_j, the *threshold value* τ *is* $f(w_{t_1^{'} q}, ..., w_{t_n^{'} q}; \overline{w_{t_1 d}}, ..., \overline{w_{t_m d}})$ and TA stops if the $k - th$ current score is equal or greater than τ. Note that, the *threshold value* represents the best possible score for unseen documents because of the monotonicity of the aggregation function and the order in the inverted lists. This means that the stopping condition guarantees that no unseen document can enter the top-k, because its best score would be smaller than the $k - th$ score.

In the TA algorithm, in order to compute the overall score for a document seen in the sequential access on a list, it should randomly access to the other lists to get the weights of the document for the other terms. A variant to the TA algorithm, called

no random access (NRA) could be used when the random access is not available to the lists. An interval of scores is computed for each document instead of the overall score. In this interval, the minimum(maximum) value represents the minimum(maximum) score of the document and is updated by traversing the lists. Note that, in this case, the documents in the current top-k are sorted by the minimum score values. Like in the TA the *threshold value* τ is computed by aggregating the last entries seen in the lists and the NRA stops when the minimum score of the $k-th$ document is greater than the *threshold value* τ and exceeds the maximum values of the candidates' scores.

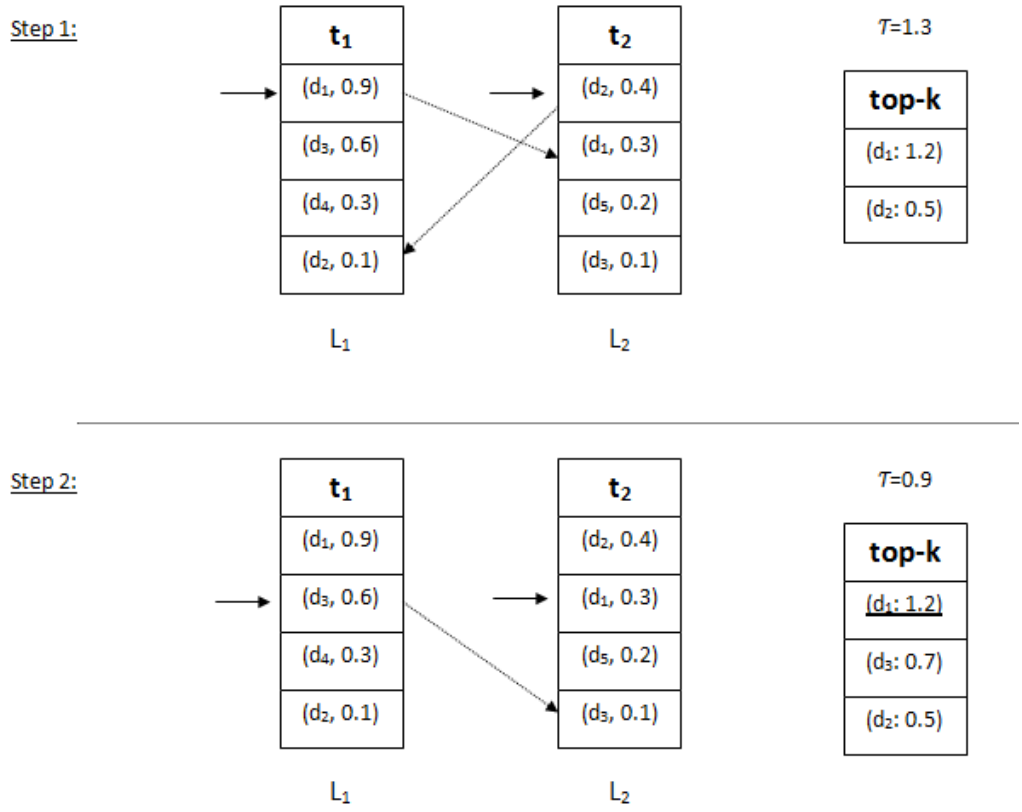


Figure 2.1: Execution example for the threshold algorithm TA

Example 1. Figure 2.1 shows the evaluation of the top-1 results of the query $q(t_1, t_2)$ with the TA algorithm. We consider that the aggregation function is the *cosine* similarity presented below. For simplicity, the query term weights are equal to 1; $w_{q,t_1} = 1, w_{q,t_2} = 1$ and we did not use here normalized weights. We consider the two lists (L_1, L_2) in the inverted index which correspond to the query terms(t_1, t_2) sorted in descending order of the documents term weights w_{d,t_1}, w_{d,t_2}.

In step 1, the documents d_1, d_2 are retrieved following a sequential access on the lists L_1, L_2 respectively. For each document, the algorithm randomly accesses the another list in order to compute the overall score. Then, the two documents can

be added to the current top$-k$ with their final scores $d_1 = 1.2, d_2 = 0.5$ and the *threshold value* is computed $\tau = 1.3$. Remember that, the the *threshold value* is obtained by applying the aggregation function to the last entry seen in each list. In this step the TA does not stop since the *threshold value* $\tau = 1.3$ is greater than the $k - th$ document score in the current top$-k$, $d_1 = 1.2$. Note that here the $k - th$ document is d_1 since $k = 1$. In step 2, TA retrieves a new candidate d_3 and the *threshold value* becomes $\tau = 0.9$. In this step, TA confirms that the document $d_1 = 1.2$ is the top-1 result for the query q and it stops since $\tau = 0.9 < d_1 = 1.2$, which means that the maximum score for any unseen document τ is below of the $k - th$ current score.

Figure 2.2 shows the evaluation of the query $q(t_1, t_2)$ using the NRA algorithm. As mentioned above, NRA only allows the sequential access to the lists and maintains an interval of scores for each candidate. In step 1, as for TA the documents d_1, d_2 are retrieved following a sequential access to the lists L_1, L_2. But here, the interval of scores is computed for each of them instead of an exact score. For example, the document score interval of d_1 is $[0.9 - 1.3]$ where its partial score in the list L_1 (the minimum score) is 0.9 and its maximum overall score is 1.3. Note that, the maximum overall score of d_1 is obtained by applying the aggregation function on its partial score in L_1, (0.9) and the maximum partial score that would have in L_2, (0.4). The documents are maintained in descending order of their minimum scores in the current top-k list. Like in TA, the *threshold value* τ equal to 1.3. In step 2, the interval scores for each candidate are updated. In this step, the document $d_1$has its exact score $[1.2 - 1.2]$. The algorithm NRA stops since the minimum score of the $k - th$ document in the current top-1 , $d_1[\underline{1.2} - 1.2]$ is greater than the *threshold value* $\tau = 0.9$ and all the maximum scores of the other documents in the current top-k list.

2.1.3 Filtering social streams

This section presents information filtering over the streams produced by the users of the social network, in this context the proposed filtering techniques benefit from the social dimension. The relationship between users introduces the social network dimension that may provide additional criteria, beyond the traditional content-based ones to measure the potential interest of information messages for users.

The increased number of users on social networks results consequently into an important increase of user-generated content. Social networks services such as Twitter[1] and Facebook[2] constantly generate millions of messages on a variety of topics every day. Without filtering these streams, it leads to an overload of information. Unlike simple text streams, social streams have new characteristics that may help in the filtering process. Users in social networks create explicit relationships like symmetric

[1] www.twitter.com
[2] www.facebook.com

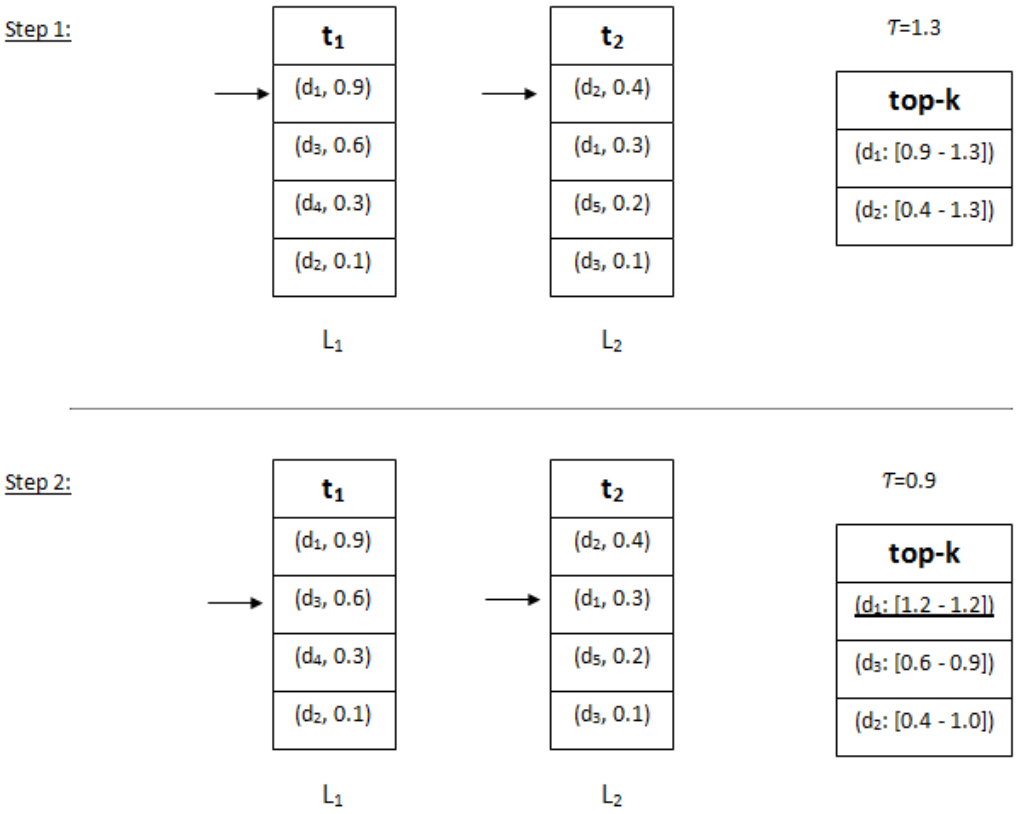


Figure 2.2: Execution example for the no random access algorithm NRA

"friendship" relations in Facebook or asymmetric "following" relations in Twitter. Thus, users receive in real-time information messages published by their friends. In addition, social networks users can evaluate content by giving feedback on messages such as likes, comments, forwards, tagging as favorite, etc. User feedback can be helpful in assessing the importance of published messages and may change the initial ranking at some point.

Content recommendation systems over social streams have widely studied in the research community [CNN+10, CCZ+12, SCZ09, KOSP11, KSJ09, PGC+10, CNC11] and in the industry. These systems use ranking models based on social network criteria. The role of these systems is to help users to be updated only with an interesting content.

[CNN+10] proposes a URL recommendation to users in Twitter by ranking the URLs. The ranking is based on text relevance score between user profile keywords and the URL description keywords. It also considers a social voting by recommending the popular URLs mentioned by most of the users.

[DFMGL12] proposes a recommendation system to recommend top-k interesting news articles to users by combining two kind of streams, news streams (from Yahoo! News) and social streams (from Twitter). News streams have a large volume of news

articles and users' social streams have social signals that can be exploited to address the problem of personalization. The relevance model is based on the profile of user's friends in the social network, on the content of user's messages in their own stream, and on the popularity of topics in the news streams and the social network.

In [CCZ+12] authors recommend useful tweets to users based on a collaborative ranking model. They consider three majors elements in the model: tweet content, user social relationships and other extra information such as publisher authority and quality of the tweet.

[PGC+10] predicts the importance of messages in Facebook feeds using support vector machine (SVM). While above works focused on recommending individual messages, [CNC11] recommends multiple messages (conversation) on Twitter, their algorithms explore three factors to discover the best ranking model: the number of messages in the conversation, the topic relevance and social relationships between users.

Typically, in the recommender systems presented above, the efficiency issue was out of the scope and real-time recommendation for million of users is not considered. These recommender systems are only proposed to provide a better relevance estimation in social network environments.

We propose a real-time recommender system that supports a large number of users in a social networks context, for each of them we recommend in real-time the best k ranked messages published by the users in the network.

In the industry, several filtering techniques and recommendation services have been proposed. Facebook uses its algorithm EdgeRank to determine messages that should appear to users when they log in. An "edge" in Facebook is any event in the network such as publishing a message, liking a message or a comment on it. Three different components are considered in EdgeRank: the strength of relationships between the edge's creator and the consumer, the edge's weight (for example, publishing a message is more important than a like) and time decay to consider the recency. At the same time, Twitter since its creation, uses a simple display strategy without any filtering technique, simply based on reverse chronological order. More recently, Twitter started to apply some filtering methods based on relevance. Also, many of third-part applications propose techniques to filter social streams such as Flipboard[3] and my6sense[4].

2.1.4 Web search with social network-aware personalization

This section explores the personalization of web search results and ranking models based on social network criteria.

[3] www.flipboard.com
[4] www.my6sense.com

Personalizing the search in order to improve the quality of results has been extensively studied in information retrieval (IR) and Web search. Search personalization adds various types of the information from the user who initiated the query when evaluating the query. Users have different preferences and interests and may need different results, even when they initiate the same queries. In the non-personalized search (user-neutral), the user who initiates the query is negligible. Thus, two different users initiate the same query, seeing the same results. While in the personalized search, user is actively involved in the search process and user information has an impact on ranking.

Several studies propose profiling models based on tracking and gathering users' preferences and interests, they also offer approaches to incorporate them into ranking models to personalize the search. In general, profiling models include users' searching histories[TSZ06], click-through analysis[JGP+05], personal bookmarks[JW03] and topics of interest[CNPK05]. Tracking implicit user's activities to create the user profile pose often the problem of privacy.

Users' social network provides a relevant public information about their needs and interests. Explicit relationships between users and interactions in the social network provide social data ideal for personalization. Users are generally more interested in content coming from friends than others in the network. On-line social services give users the possibility to express their opinions by publishing content and also by evaluating content through feedback such as likes, comments, forwards, tagging as favorite, etc. User profiles created from the public social data would be appropriate to express user's interests and do not violate the privacy.

In the following, we present works that focused on personalizing search results and improving the textual search on the Web using social data. Several ranking models to combine social data with textual relevance rankings have been proposed to improve the accuracy and quality of query results [KS12, YLL10, GCK+10, CZG+09, MGD06].

[CZG+09] investigates personalized search over "social" data collected from Web 2.0 applications, such as social networks, blogs, forums, etc., based on the user's social relationships. They re-rank the search results by considering the relationships between users that belong to the searcher's social network and the results. Thus, objects that are strongly related to the "close" users in the searcher's social network, are highly ranked. The relationships strength with the objects in the results depends on its type, which could be authorship, tagging, or commenting. They experiment with different social networks of the user either through explicit connections or implicit connections through social activities (e.g., tagging with the same tag, liking the same object). They show that personalization based on the social network factors significantly outperforms non-personalized social research and other personalization strategies.

[KS12] proposes a scoring model to calculate socio-textual relevance between web documents and users by utilizing social data available in their social network. The

proposed model considers several social parameters including relationships between users, importance of users in the network and actions that users perform on web documents. Then, they linearly combine the social parameters with the textual relevance model (tf-idf).

[YLL10] focuses on efficient top-k search on Web documents and proposes two processing strategies: textual relevance TR-based search and social influence SI-based search. The scoring model combines two factors, textual relevance (TR) and social influence (SI) to determine the relevance of documents relative to users' queries. SI relevance is represented by the distance between two users in the social graph and the tf-idf model is used to compute TR relevance. The two strategies operate in a pipeline fashion. TR-based strategy firstly fetches in descending order of TR score all documents with non-zero TR score, then for each retrieved document, it traverses the social network to determine SI score between the querying user and document publisher, finally it computes the total score. SI-based strategy firstly traverses the social network and get users in descending order of SI, then for each document, the total score is computed.

[GCK+10] proposes a social network-aware ranking framework that considers both text relevance and social relevance. Text relevance is based on popular models like tf-idf or BM25. Social relevance relies on the relationships between document owners and the querying user in the social network. The relationships between two users is modeled by a similarity function which is based on the structural information of the social network. For example, the more common neighbors between two users more those users are similar. They propose an algorithm MAS to compute the similarity functions. They focus on how to efficiently compute these similarity functions over large social networks with millions of users.

The studies in this section focus on discovering different social signals and how to integrate them with traditional content-based ranking models to improve ranking relevance and improve search results. These methods are designed for searching web documents, but their complexity prevents their use for streaming data with continuous top-k processing. Our work aims at personalizing the query results on the basis of criteria issued from the user's social network. We propose a rich score model, including social network components, providing a good compromise between expressiveness and complexity for continuous top-k processing of information streams.

We can classify the social criteria used in the scoring model for search personalization on the Web into two components:

1. *User based factors,* measuring the importance of users and of their relationships in the social network. In this category, we can consider two different parametrs, the global importance of the user in the social network and the relative importance between the users. In most cases user-based importance is measured on the social network graph, by evaluating e.g. node centrality and distance between nodes.

2. *Interaction based factors,* measuring the importance of documents by the reaction they provoked, expressed through actions of other users on that document. Depending on the social network context, current actions may be likes, comments, forwards, tagging as favorite, etc.

2.1.5 Processing top-k queries in a social network context

In this section, we present the works that addressed the problem of processing *snapshot top-k query* in a social network context, particularly in social tagging networks, with a focus on efficiency issues. In this context, top-k retrieval algorithms considering social criteria have been proposed in [MC13, SCK+08, YLAY09].

Social tagging networks, such as Delicious[5] or Flickr[6], are a particular case of social network, where users can publish items and also add some keywords (tags) to items published by other users. Items in social tagging networks could be images, videos or text. Collective human tagging is a significant feature for describing and classifying the content and more importantly make it search-able. Browsing the information by tags is an effective manner to access content. In this social environment, users may also be interested in querying the content and retrieving the most relevant results. "Tags" added manually by users make these social networks rich in both textual and social factors, it is therefore interesting to apply a combined textual-social top-k query in such social networks where items that are tagged by the querying user's "friends" should be ranked higher. Traversing all users and testing their items for ranking is a costly operation. In such context, efficient processing for snapshot top-k queries is a challenging task due to a large number of items in the social network. In the following, we describe the studies that address the problem of processing *snapshot top-k query* in social tagging networks.

[YBLS08] proposes network-aware search in collaborative tagging sites and defines the problem of efficiently processing top-k queries. Authors consider only a subset of users that contribute to the social relevance score, those who have a direct relationship with the querying user. The relevance score of an item i for a user u w.r.t a tag t_j is defined as a montone function of the number of users in the social network of u who tagged i with tag t_j. They develop generalizations of top-k processing algorithms (NRA[FLN01] and TA[Fag02]) to incorporate network-aware search.

In [SCK+08], the social relevance is considered under the general interpretation, users in the whole network could contribute to it. The social scoring of item i for a user u w.r.t a tag t includes the importance of a user v relative to u and the importance of the tag t for the user v w.r.t the item i. All users v_j that have a path with u in the social graph are considered in the social relevance. Authors maintain an index composed of three types of lists. Tag-item lists, for each tag t an inverted

[5] www.del.icio.us/
[6] www.flickr.com/

list of items tagged by the tag t. User-tag-item lists, for each user u and each tag t, they maintain an inverted list of items tagged by u with t. User-user lists, for each user u, they maintain a sorted list containing other users v_j with their importance relative to u. They propose disk-based algorithm ContextMerge which follow the principle of top-k threshold algorithm TA [Fag02] over sorted lists. At each step, ContextMerge chooses either to test the items retrieved from User-tag-item lists (items tagged by the closest unseen user) or from Tag-item lists.

In [MC13], the social relevance is similar to the one of [SCK+08] and also considers the users in the whole network but the similarity between users is not pre-computed as in [SCK+08]. They propose a technique to compute the social relevance on-the-fly for a large family of functions for similarity computation in a social network. This is done based on the generalizations of Dijkstra's shortest paths algorithm [Dij59]. Then, an exact algorithm for top-k queries using a memory-based index is proposed. In order to improve the execution time, authors explore some approximate techniques in the computation of top-k results. Particularly, they use approximation for the on-the-fly computation of the similarity between users in the social network.

we present the works that addressed the problem of processing *snapshot top-k query* processing in a social network context,

introduces techniques and approaches for processing *snapshot top-k query* that use ranking models based on social networking criteria

These algorithms propose an efficient *snapshot top-k query processing* in a social network context using a specific index structure for a large number of items. We address *continuous top-k processing* in a social network context, with top-k sets recomputed at each publication or feedback event, for all the users in the social network.

2.1.6 Real-time search

The previous section presented the works focused on the efficient processing of a *snapshot top-k query* in a large social network. This section presents the studies focused on processing top-k queries in a multi-query environment. This environment receives a stream of messages and queries. The objective here is to handle the stream of messages by indexing them in real-time and to efficiently evaluate the queries over the messages. To process the *continuous top-k queries*, the *static approach* is considered where the query results are updated by running periodic snapshot queries on the messages as in [BGL+12]. Note that, in our solution we adopt the continuous processing approach that indexes the queries and avoids the indexing of the stream messages. Furthermore, in real-time search, new queries are applied to existing messages and new messages, while in our work new queries are only interested in new messages.

In social networks services and particularly in mircoblogging such as Twitter, users expect real-time search, i.e information should be immediately available in search

results in order of few seconds after the publication. Thus, information should be indexed efficiently and this is difficult given the throughput of message publication. Conventional indexes proposed in the information retrieval community are not designed to support very high update rates since in most cases these indexes are based on inverted lists where their entries are sorted by a document's partial score or by the document identifier so updating with new documents may be expensive. To address this challenge many works have addressed the updates overhead problem [BGL+12, XXWL13, CLOW11].

Twitter's Search service supports real-time search using the Earlybird index [BGL+12]. It uses the conventional inverted lists to index messages (tweets), and maintains tweets in chronological order. This is highly efficient for tweet insertions, as a new incoming tweet do not affect the order of entries, and is appended at the tail of the lists. Earlybird is organized in two types of indexes: a read-only index only for query evaluation and a write-friendly index that supports both efficient tweet indexing (write) and query evaluation (read). To manage the consistency in the write-friendly index, they adopt a multi-thread environment where concurrent queries are handled by multiple threads and indexing tweets is done by a single thread. The synchronization between read and write threads is guaranteed by the JVM[7]. They use a scoring function that combines different signals: static signals, *interaction based factors* and *user based factors*. Note that the text relevance score is not considered in their scoring function, Earlybird supports boolean queries and phrase queries. The recency is ensured by traversing lists in chronological order. However, the scoring function and the query evaluation algorithms are not presented at all in this study.

Unlike in [BGL+12], [XXWL13] uses a more complex scoring function that considers three components: the global importance of a message, the text relevance and the time dimension. The time is modeled by an exponential decay function (more details on decay functions are presented in section 2.1.7.1) and the text relevance score depends on tf-idf model. Unlike in the Earlybird wher new queries are applied to existing messages and new messages, here queries are only applied to existing messages. To enable a real-time search using the different factors of the scoring function, they propose the Log-Structured Inverted Indices (LSII). LSSI consists of a sequences of inverted indexes $I_0, I_1, ..., I_m$ where each of them maintains a disjoint set of documents. The first index I_0 is identical to Earlybird, it maintains a fixed number of newly arrived documents. The other indexes $I_1, ..., I_m$ maintain older documents where each term t maps to three lists corresponding to scoring function components, the first list sort documents in descending order of the global importance of the document, the second sort them in descending order of the term's weight and the last list sorted in descending order of timestamp. LSSI can efficiently handle updates as new documents are only inserted into I_0 in chronological order, an efficient merge strategy is applied to flush documents from one index to another. LSSI also evaluates efficiently queries as follows. At query evaluation, LSSI first

[7]Java virtual machine

checks I_0 to identify the k documents with the highest score. Then, it applies the threshold Algorithm (TA) [Fag02] over the sorted lists corresponding to the query terms in all other indexes $I_1, ..., I_m$. The documents in $I_1, ..., I_m$ indexes have smaller scores compared to those in I_0 because the time decay effect, thus TA tends to terminate early. LSSI manages also the dynamic score of a document following the change of the global importance over time. They also assume the system works in multi-thread environment where there are a reader thread that processes queries, a writer thread that handles documents insertion, and multiple merger threads that deal with index mergers.

We can remark that the above approaches focus on efficiently indexing the incoming documents and make them search-able immediately. In addition, they focus on efficiently evaluating queries on this dynamic index of messages. In such solutions, the continuous top-k query is managed by the *static* approach as in Earlybird[BGL$^+$12]. As mentioned in the beginning of this section, we focus on continuous query processing reacting to new messages and to other events (interactions), in order to incrementally maintain the lists of top-k messages.

2.1.7 Continuous processing of top-k queries over text streams

Our work focus on the *continuous processing* of top-k queries over information streams in the social networks. Social networks have millions, up to billions of users and streams messages. The main challenge is how to design and implement an efficient processing at a very large scale in terms of the number of the queries and messages. In this context, we consider a complex scoring model including textual, social and time criteria.

In this section, we present the scoring functions considered in the literature and describe the techniques for modeling time dimension. We present the algorithms that deal with continuous top-k queries evaluation in publish-subscribe systems.

2.1.7.1 Scoring functions

In this section, we describe the scoring functions considered in the continuous top-k queries on information streams. As mentioned above, continuous processing employs simpler scoring functions than the *static approach*.

The scoring function measures the importance of a message w.r.t. a query. The choice of the scoring function has a significant impact on the efficiency and effectiveness of continuous top-k systems. A simple scoring function can speed up the query evaluation, but it could have a negative impact on the results quality. On other hand, a rich (complex) scoring function can yield better results quality with complexity in the query evaluation.

Text relevance

Since textual content is characteristic to information streams, content-based queries are usually based on sets of terms of interest, and the importance of a message is evaluated from an information retrieval (IR) perspective, as the relevance of the text message to the query, based on weighting models such as tf-idf [SB88] . Typically, vector space model(VSM) is used to represent both messages and queries as a vector of terms with a tf-idf weight associated to each term. Then, the text relevance score is computed using a similarity function such as the cosine function. Given a message m and a query q and by considering normalized weights, the cosine similarity function between m and q is defined by the following:

$$cos(q, m) = \sum_{t \in q \cap m} w(q, t).w(m, t)$$

Where $w(q, t)$ and $w(m, t)$ are the term weights of the term t in q and m, respectively.

Our work uses the tf-idf model with the cosine function to measure the text relevance score, knowing that other weighting models and similarity functions could be used in our solution as explained in subsection 3.2.2.

Social relevance

The social scoring is lacking in most of the works on continuous top-k queries over text stream. To the best of our knowledge, the only work on continuous top-k processing for information streams including a social network component in its score model is [VAC12], but this is limited to the simplest component, a global, query independent importance of each message. Recently [Vou15] reported an extension of this work including user feedback in the score model, which can be assimilated to our interaction score components in subsection 3.1.2. However, the *user based factors* that measure the importance of relationships between the users in the network are not taken into consideration. In our ranking model, the social relevance score considers *interaction based factors* and the *user based factors* with its two different parametrs: the global importance of the user and the importance of relationships between the users.

It is important to note that including in the scoring function the *user based factors*, measuring the strength of relationships between the users in the network is a challenge given the size of the network. The on-the-fly computation of this parameter as in the snapshot top-k query [MC13] is impractical in the continuous top-k scenario. In our implementation, we followed [SCK+08] by pre-computing this parameter. The *Interaction based factors* add another significant challenge to the continuous top-k scenario because the actions on the messages update (increase) their scores. Thus, it is possible that a message that has not entered into a user's top-k, will enter in the future because of the actions.

Time factor

Another important component of the scoring model for continuous top-k queries is time. In the following, we describe how the time dimension is incorporated into the scoring model.

Streaming data is usually characterized by a temporal dimension. All data stream items are accompanied by an associated timestamp representing their creation or update time. The time dimension plays an important role for the quality of the results in information stream systems. Intuitively, more recent items are considered more relevant for query results. Incorporating data recency in ranking algorithm should allow them to cope with data quality and also scale well within streaming system. Pub-sub systems in the literature have considered two models for handling the time aspect in the information stream: *the sliding window model* and *decay functions*.

Sliding window model

Many studies of continuous queries over text streams [HMA10, MBP06, HMA12]have used this model for handling the time dimension. This model considers a simple scenario based on keeping in a window only the most recent items of the stream. Therefore, query results contain only the valid items, i.e. those in the window. The semantic of the sliding window is simple, the newly arrived items are inserted into the window and the oldest ones are dropped out from the window. Controlling the number of items present in the window can be done either by count or time. In count-based scenarios, the window only maintains a pre-defined number N of items. In time-based scenarios, the window holds items that occur within a particular time interval, e.g items in the last 2 hours. The sliding window model is a simple model to represent the temporal dimension, it allows a direct incorporation into the ranking algorithms, since time is not part of the scoring functions. But, it generates an extra processing effort when some items expire (are dropped out the window). Some queries, for which expired items are part of their results, need to be re-evaluated.

Decay Functions

In the decay function model, time is part of the scoring function. Time is explicitly handled by considering the age of items. The decay function continuously decreases, so the scores of items decrease. Therefore, the newly arrived items have greater chances to be ranked higher. Recent studies in continuous top-k queries over text streams [SGFJ13, VAC12, CC15] have considered decay functions to model the temporal dimension.

Formally, a decay function is defined by a monotone decreasing function $TD : \mathbb{R}_+ \rightarrow [0, 1]$ with $TD(0) = 1$. For an item i published at time t^i, the score of item i for a query q is computed by a scoring function $score(i, q)$ and the variation in time of the score of i for q is expressed by the time-dependent scoring function $tscore(i, q, t)$

such that for any moment $t \geq t^i$:

$$tscore(i, u, t) = score(i, u) \cdot TD(t - t^i) \tag{2.1}$$

Generally, only order-preserving decay functions are considered, i.e. functions that preserve in time the relative order of items scores. That means that if $tscore(i, q, t) < tscore(i', q, t)$ then $tscore(i, q, t') < tscore(i', q, t')$ for any another moment t'. The order-preserving decay prevents the reordering of the items in top-k results in time. For instance, a linear or exponential decay function could be considerd in this case.

The use of time in the two models is different. In the sliding window the representation of time is implicit, all items present in the window are equivalent in terms of time. The difference between the newer and older items only appear once an item expires i.e its score suddenly becomes zero. Whereas, in the decay functions, the representation of time is explicit as a part of the scoring function. The scores of items continuously decrease over time.

On the other hand, unlike in the sliding window, decay functions do not generate an extra processing effort since items do not expire. But, due to the continuous decrease of scores as time passes, this needs to frequently recalculate scores of all items.

Note that, the sliding window could be seen as a particular case of the decay function, but it is not an order-preserving decay function. $TSW(i) = \begin{cases} 1 & i \in window \\ 0 & i \notin window \end{cases}$

In the present work, we consider the time decay model. To avoid the re-computation of scores we adopt the dual approach of time-bonus functions inspired from [CSSX09]. More details about this approach are presented in subsection 3.1.2.

2.1.7.2 Continuous top-k algorithms in publish-subscribe systems

We describe in details the closest algorithms to our work, that consider continuous processing for the continuous top-k queries over information streams, and deal with a large number of subscription queries and publications in a pub/sub environment. The ranking model considers text relevance and time (recency) in most of these algorithms. These algorithms provide efficient processing by processing messages on the fly and by using data structures to index queries and to efficiently determine the impacted queries for each message.

[PvA08] is the first work that proposes a new publish-subscribe model (top-k/w publish- subscribe) that continuously maintains for each subscription the best k publications in a predefined sliding time window w. They propose a probabilistic computation for top-k/w queries. The proposed solution does not expect a large

number of subscriptions as it evaluates the incoming publication with every subscription.

As opposed to the above work, all the following works adopt data structures for subscription queries to enable the efficient processing.

[MP11] proposes a solution for top-k/w publish-subscribe over text message streams based on classical tf-idf cosine similarity. It uses a sliding window to model the temporal dimension. The proposed data structure used in their system is composed of a single list of the valid documents D which belong to the sliding window and two inverted indexes. The first index for documents D, is composed of lists for each term t , containing the document d that has t, sorted in descending order of the term's weight $w_{t,d}$. The second inverted index is for subscription queries Q where the lists are sorted by the *threshold value* $\theta_{q,t}$ for the query q that includes t.

When a query q is submitted to the system, the TA algorithm is applied on the first index structure for the initial computation of top-k lists. This will be efficient as TA does not retrieve all entries in the lists but it benefits from the ordering of the lists to early stopping while traversing them. In this phase, the top-k list for the submitted query q is computed. Then for each term t in the query q, the threshold value $\theta_{q,t}$ *is initiated by the document term weight* $w_{d,t}$ *which is the last retrieved entry in the list of the term* t *after the TA algorithm stops.*

When a new document d arrives, it is inserted in the corresponding inverted lists in the document index. Then, for each term t in the document d, they identify in the second index all queries Q_i where $\theta_{Q_i,t} \prec w_{d,t}$; these queries are candidates that may be affected by d. All the other queries Q_j where for every term t in the Q_j $\theta_{Q_j,t} \succ w_{d,t}$ are not affected by d. However, since documents are indexed, a high arrival rate results here in expensive index updates.

[HMA10] also tackles top-k/w publish-subscribe on text information streams and proposes the COL-Filter algorithm and an improved variant POL-Filter. It also considers the sliding window technique to manage time aspects and the TA algorithm in the core of their two proposed variants. Unlike the previous work, the messages are not maintained in the data structure.

COL-Filter only indexes subscription queries and uses a score-oriented order for the inverted lists. More precisely, a list for a query term t indexes queries q containing t , ordered by the ratio between the importance of the term t in q, $w_{q,t}$ and the current score of the $k-th$ document in the top-k list of q, $S_{min}(q)$, in other words in the descending order of $\frac{w_{q,t}}{S_{min}(q)}$. Note that $S_{min}(q)$ is the threshold for new documents to enter the top-k list. This allows efficient top-k processing by using the TA algorithm on the index lists since queries in the inverted lists have a good locality following the ratio $\frac{w_{q,t}}{S_{min}(q)}$. Queries with a small value of $S_{min}(q)$ or an important value of $w_{q,t}$ will be ahead in the lists. This enables TA to early find all good candidates ahead in the lists for top-k update.

When a new document d arrives in the system, like in TA they traverse the corresponding inverted lists in round robin fashion. They evaluate all retrieved queries

Q_i with the document d using the similarity function score and if the score is greater than $S_{min}(Q_i)$, the document d becomes a result in the top-k list of Q_i. To avoid retrieving all queries which appear in an inverted list of a term t in the document d, they propose a stopping condition to be ckecked while traversing the inverted lists. They prove that this stopping condition is correct: all queries for which a new incoming documents serves as a top-k result appeared in the inverted lists before this stopping condition.

This method suffers from a relatively high number of updates for maintaining the inverted lists subsequent to $k-th$ score changes. Message exit from the time sliding window also results in updates to the top-k results.

While the previous works use the inverted list to index queries and evaluate documents, [RCCT14] indexes queries by a graph index based on a covering relationship between subscription queries. They proposes a strategy for sharing effort among queries in the top-k computation process. The covering relationship is defined based on the query terms: a query q covers q' if the terms in q are superset of terms in q'. Based on this covering relationship, a directed graph GIS is created where nodes represent queries and a directed link indicates that the source query covers the destination query. A directory containing all distinct terms appearing in queries is also maintained in the system.

Evaluating an incoming document d is done by checking the directory to retrieve root queries for each term in the document. Then, the document is evaluated with all queries by traversing the graph starting with those retrieved root queries. Following the definition of the covering relationship, they derive a number of lemmas that define interesting properties between the relevance score of d to the covering query q and to the covered query q'. The proposed lemmas help save the evaluation cost.

However, the proposed lemmas are not adapted to the extension with more complicated score functions including social network criteria since the covering relationship is defined on the basis of the relationship between the queries terms. We think that the proposed lemmas could be extended to more complicated scoring functions that rely only on the text component.

[SGFJ13] proposes an adaptation of two IR top-k retrieval strategies to information streams: the document-at-a-time (DAAT) algorithm WAND [BCH$^+$03] and the term-at-a-time (TAAT) algorithm of Buckley and Lewit [Cal96]. Instead of time sliding windows, a continuous order-preserving decay function is proposed to handle time-dependent scoring, which eliminates the problem of top-k recomputing upon message expiration. Unlike the above index structures, the queries are indexed two times in two different data structures. The first data structure indexes the queries based on their identifiers. For each term t, an inverted list containing all queries that contain t and sorted in the ascending order of queries identifiers. The second one is a tree-based index, for each term t a balanced binary tree where leafs represent the queries containing the term t and maintaining the corresponding $S_{min}(q)$, the minimal $k-th$ document score in the top-k list. Each node in the

tree also stores the minimum value of $S_{min}(q)$ for all queries Q_i in its sub-trees i.e, $min(S_{min}(Q_i))$.

For the adaptation of Buckley and Lewit, when a new document arrives d, the lists corresponding to each document term t are sorted in the descending order of the maximal partial score of t in the list. The proposed algorithm processes the lists sequentially and examines an early-termination condition to eventually skip the remaining lists. When the upper bound on the score of d is below the $S_{min}(q)$ for every q in the list L_j, then the list L_j can be skipped. In this condition, the $S_{min}(q)$ for every q in the list L_j could be very small or be zero in some cases, making the skipping strategy ineffective. They optimize the skipping strategy and skip a segment of a list instead of the whole list. They use the tree to efficiently look for any query q in the list that violates the above condition i.e, has $S_{min}(q)$ smaller than the upper bound score of d. Since the nodes in the tree store the minimum value of $S_{min}(q)$ for all queries Q_i in its sub-trees, then these queries in the sub-trees can be skipped if their root node holds the above condition.

Note that when top-k changes, only the tree-based index should be updated. Similar to our work the decay function is considered to model the recency, but they consider the exponential decay function, which has the property to be easily transformed into an equivalent bonus function..

Unlike the above approaches considering text information streams with monotonic and homogeneous scoring functions, [VAC12] introduces a more general scoring function by combining item importance score with query-document relevance score. This results in non homogeneous scoring functions, where methods proposed by the approaches above are not applicable. Like in[SGFJ13], time-dependent scoring is handled through decay functions. They use a two-dimensional inverted query indexing scheme and explore efficient score bounds with drastic pruning of the search space. Queries are indexed in spatial indexes with two dimensions, the minimal score of a query $S_{min}(q)$-dimension and the term weight $w_{q,t}$-dimension. More precisely, each query q is maintained in the inverted grid for each of its terms $t \in q$ by the $S_{min}(q)$ and $w_{q,t}$. Based on this index, they define three linear upper bound conditions spatially determining for each incoming document d and for each term $t \in d$ a subset of candidates queries whose top-k results is potentially impacted by d. Then for each candidate query, they compute the total score and check if its top-k list is updated by d. For all queries updated by d, their minimal score $S_{min}(q)$ is increased and the queries are moved to a new position in all inverted grids corresponding to the queries terms. They propose four data structures in the design of their index that take into account this moving behavior of queries following top-k update.

All these continuous top-k techniques are hardly extensible to include social network criteria. A major drawback is the need of many index updates, since they include the value of the k-th score (having frequent changes) into each index dimension. Moreover, they consider short queries (a few terms), while social network environments come with implicit subscription queries based on user profiles (long queries).

Since the number of updates grows with the size of the query and with the number of dimensions (that increases when introducing social network criteria), these techniques are not adapted to the social network context. Our method separates the k-th score from the other dimensions in the index, thus minimizing impact of updates and facilitating the extension with new social network dimensions.

2.2 Diversity-Aware top-k query processing over information streams

The problem of query results diversity has received much attention for many years in different domains, databases [LSC09, DFZN10], recommendation systems [ZMKL05] and information retrieval [RBS10, CKC+08, LTG09]. Results diversification is a way to increase user satisfaction and to cover different aspects of the query. For that purpose, systems should combien diversity and relevance in search results. For a given query, the results set must include the most relevant documents and at the same time as diverse as possible. Several approaches have been proposed to define semantics of diversification [GS09, CKC+08].

In general, query result diversification can be categorized into content-based diversification and intent-based diversification. Content-based diversification focuses on increasing dis-similarity between documents in search result. Intent-based diversification explicitly interprets all possible intents (topics) of the user's query by using external information. Our work belongs to the *content-based diversification* approach.

Intent-based diversification explicitly interprets all possible intents (topics) of user's query by using external information, like taxonomies, query logs, etc., then it returns objects that covers as many topics of the query as possible [AGHI09].

Content-based diversification focuses on increasing dis-similarity between documents in search result and avoiding information redundancy. This is achieved by filtering out documents that are similar to existing documents in the search results. In general, a distance (dis-similarity) function is used to measure the distance between documents.

Many studies for results diversification [CC15, MSN11, GS09] rely on a bi-criteria objective function (originally proposed in [GS09]) that combines in a linear way the relevance of documents to the query and the distance between documents. The most popular objective functions lead to *Max-Sum diversification* and *Max-Min diversification,* presented below.

Let $O = \{o_1, o_2, ..., o_n\}$ be a set of n relevant objects for a query q. Let $S_k \subseteq O$ be a subset of k objects where $k \leq n$. Let also the relevance of an object $o \in O$ to the query q be defined by the function $R(q, o)$, where a higher value indicates that o is more relevant to q. The diversity of a result set S_k is defined by aggregating the

dis-similarity between every two objects $o_i, o_j \in S_k$ and the dis-similarity is defined by a distance function $D(o_i, o_j)$, where a low value means the two objects are more similar.

Typically, the trade-off between the relevance and the diversity is modeled by an objective function. The objective function is defined over a set of objects, and takes into account both the relevance and the diversity. Let us consider an objective function $F(P(O), R, D)$, where $P(O)$ is the power set of O, R is the relevance function and D is the distance function.

The *content-based* diversity problem focuses on identifying the optimal set S_k^*, by maximizing the objective function as follows:

$$S_k^* = \underset{\substack{S_k \in P(O) \\ |S_k = k|}}{argmax} F(q, S_k, R(.), D(.,.))$$

As presented in [GS09] finding the optimal subset S_k^* by solving the above problem is NP-hard problem for various objective functions. Thus, several heuristics techniques have been proposed in the literature to find the approximate subset S_k^*.

In the following, we present the two most popular objective functions, *Max-sum* and *Max-min*. Then, we present some of the proposed algorithms in literature to approximately compute the subset S_k^*.

The *Max-sum diversification* function is based on the sum of relevance and dis-similarity measures of the top-k set and is defined as follows:

$$F_{sum}(q, S_k) = \lambda \sum_{o \in S_k} R(q, o) + (1 - \lambda) \frac{2}{k-1} \sum_{\substack{o_i, o_j \in S_k \\ i<j}} D(o_i, o_j)$$

The first term sums the relevance score for each object in the returned set. The second term sums the distance of all pairs (o_i, o_j) in the set where $i < j$. $\lambda \in [0, 1]$ is a trade-off parameter between relevance and diversity. To compensate the fact that there are k values in the relevance sum versus $k * (k-1)/2$ values in the distance sum, the second one is penalized by a factor of $2/(k-1)$. The diversity problem that uses the Max-sum objective function aims to maximize the weighted sum of relevance and dis-similarity for the selected set.

The *Max-min diversification* only considers the minimum relevance and the minimum distance and is defined as follows:

$$F_{min}(q, S_k) = \lambda \underset{o \in S_k}{min} R(q, o) + (1 - \lambda) \underset{o_i, o_j \in S_k}{min} D(o_i, o_j)$$

Here the diversity problem aims to maximize the minimum relevance and the minimum dis-similarity for the selected set.

In the following, we present the algorithms of the state of the art for computing a diverse top-k result set over a collection of static objects. Next, we present the difficulties that arise when computing the diverse top-k result sets over streaming data. Finally, we describe the results diversification problem in a publish-subscribe system for a large number of queries.

2.2.1 Query results diversification

Since the divesification problem in its general form is NP-hard, several works have proposed heuristic techniques to find the near-optimal set S_k among all relevant objects O. In general, they employe *greedy* or *interchange* heuristics for computing the diverse top-k resutls lists. With the *greedy* heuristics, the algorithm selects the optimal item according to a heuristic technique at each time. With the *interchange* heuristics, the algorithm establishes k resutls according to some simple strategy, then it tries to improve it by an interchange operation between an item in the current result S_k and one in O.

In the following, we present approaches that use the *greedy* heuristics.

[CG98], proposes a greedy approach, the Maximal Marginal Relevance (MMR) algorithm that iteratively constructs the set S_k. The MMR algorithm ranks the objects using the $mmr(o_i)$ function that coressponds to the *Max-sum diversification* as follows:

$$mmr(o_i) = \lambda R(q, o_i) + \frac{(1-\lambda)}{|S_k|} \sum_{o_j \in S_k} D(o_i, o_j)$$

At each iteration, MMR selects the most relevant object to the query (the first term), and at the same time the most distant to the objects in the current S_k (the second term).

[VRB+11] proposes two methods, named Greedy Marginal Contribution (GMC) and Greedy Randomized with Neighborhood Expansion (GNE) to find the set S_k. In the two methods, they employ the Maximum Marginal Contribution (mmc) function to rank the objects. Compared to the mmr function, mmc considers not only the relevance and the distance to the objects aleardy selected, but also the distance to the remaining objects in O that could be inserted into S_k.

$$mmc(o_i) = \lambda R(q, o_i) + \frac{(1-\lambda)}{k-1} \left(\sum_{o_j \in S_{p-1}} D(o_i, o_j) + \sum_{\substack{l=1 \\ o_j \in O - S_{p-1}}}^{l \leq k-p} D^l(o_i, o_j) \right)$$

Here $1 \leq p \leq k$, S_{p-1} is the current result set of size $p-1$ and $D^l(o_i, o_j)$ is the l^{th} largest D value in $D^l(s_i, s_j) : s_j \in O - S_{p-1}$.

Note that, in *mmr*, when the current result set S_0 is empty, the objects are only ranked based on the relevance, while *mmc* considers the relevace and the diversity (the third term).

The difference between the GMC and GNE is how to choose the objects to be included into S_k after ranking with the *mmc* function. In GMC, the object with the highest *mmc* value is always included in the result set, while in GNE, a random technique is used to choose the objects.

[GS09] presents the Max-Sum Dispersion (MSD) method which is based on the *Max-Sum Dispersion Problem* described in [HRT97]. At each iteration, MSD selects a pair of objects that are relevant to the query and are distant one from each other.

$$msd(o_i, o_j) = (1-\lambda)(R(q, o_i) + R(q, o_j)) + 2\lambda D(o_i, o_j)$$

[YLAY09] is based on an *interchange* approach, the Swap algorithm which is composed of two phases. The first phase initializes the result set S_k with the k most relevant objects in O, follwing the $R(q, o)$ values. In the second phase, at each iteration an object among the remaning objects in O with lower relevance score is replaced with each object in the current set S_k, and if it increases the overall *Max-sum* objective function, then it takes place permanently.

In [vLGOvZ09], they try to solve the diversity problem by clustering the objects in k clusters, then one object from each cluster is selected to be result in the result S_k. They use the $k-$medoid algorithm to generate the clusters from objects in O using the distance function $D(.,.)$.

While the above approaches could be applied in all domains and typically do not consider the efficiency issues, [AK11] proposes an algorithm to find the diverisfied top-k set for keyword queries in the information retrieval (IR) domain and focuses on the query processing efficiency. They develop an efficient algorithm based on the threshold algorithm TA [FLN01] for a large number of documents. The proposed algorithm indexes the documents in inverted lists sorted by terms weights, and it also uses an additional data structure maintaining information about the diversity. They define a usefulness score of a document as a probablity function that combines relevance and diversity. The threshold value is the upper bound usefulness score for the non-seen documents. To obtain increasingly tighter threshold values, they perform a sequential access on the inverted lists to bound the relevance and define new data accesses on the diversity data structure to bound the diversity.

The proposed algorithms for results diversification are applied over *static* collections of objects. All these studies differ from our context where we deal with a large number of queries over continuous data in a pub/sub system.

2.2.2 Diversification for top-k queries over streaming data

Our proposal for results diversification aims to continuously keep diversified top-k result lists over streams of messages. The algorithms described above suppose that the collections of objects are static and the diversified top-k lists do not change. These algorithms are not adapted to the case of dynamic collections of objects. Recomputing the diversified top-k lists over the updated collections is time-consuming and incremental approaches must be explored. In addition, to allow an efficient processing, all objects should be holded in the main memory, which is impossible when the system has a large number of streamed objects. Thus, it is necessary to redefine the diversity problem on a subset of data, usually sliding windows over the stream. In the stream data, objects come with a temporal dimension represented by their creation timestamp. The new definition of the diversity over continuous data should consider the time in order to provide relevant, diverse, and "recent" objects.

The continuous version of the diversification problem has been addressed in several works [MSN11, DP12, DP09], where the diversified top-k lists is computed over continuous data. In general, these works employ *greedy*, *interchange* or *index-based* approaches.

[MSN11] proposes an incremental approach to maintain an approximate diversified top-k set, and is applicable to Max-Sum and Max-Min objective functions in the context of continuous data. It processes the input as a stream of items and continuously maintains a diversified top-k set at the arrival of a new item. The proposed algorithm uses the *interchange* heuristics by replacing each item in the current result set with the new item and maintains the set that maximizes the objective function. Similarly to our approach, recency is considered as a decay function included in the relevance score. We demonstrate in the results section that this approach does not scale to a large number of queries.

[DP09] Drosou and Pitoura tackle the results diversification problem in the context of publish-subscribe systems. They propose the SCG *greedy* algorithm that considers the Max-sum objective function and computes the k most diverse and relevant items over subsets of data. They apply the diversification over sliding windows that keep a fixed number of the most recent items. To replace the expired items because of the sliding window effect, the SCG algorithm selects the optimal items in the window that maximize the Max-sum objective function.

Drosou and Pitoura in [DP12] also propose an efficient algorithm for the Max-min diversification problem based on *cover trees*. They also adopt a sliding-window model where the diversified top-k set is computed over sliding windows. In a different way that for their study in [DP09], they propose an index-based approach that allows the incremental evaluation of the diversified sets to reflect object updates. The proposed data structure supports efficient insertions and deletion of objects that appear in the sliding window. They focus on Max-min objective function in which the relevance score is negligible, all relevant objects are indexed in a cover-tree which has been proposed for approximate nearest-neighbor search in [BKL06].

Cover trees are constructed based on the distance between objects. The cover tree is a hierarchy of levels where each level is a "cover" for all levels below. The lowest level contains all objects in the sliding window. Objects at higher levels of the tree are more distant from one another than objects on the lower levels of the tree. Thus, the most diverse objects could be retrieved from the higher levels of the tree.

[CACH14] proposes a novel diversity model in microblogging posts based on a threshold over diversity dimensions like time or sentiment polarity. The objective is to select the smallest subset of posts that match the keywords query. The selected subset should cover all the non-selected posts with respect to the diversity dimensions. This study is fundamentally different from all above works, since the diversity model does not depend on inter-items similarity metrics and bi-criteria objective functions.

However, the above studies can not work well in the context of pub/sub systems with a large number of queries. These works do not investigate query filtering techniques, but evaluate the new item with all the queries in the system.

2.2.3 Diversification in Publish/Subscribe system

The closest work to our study in this domain is [CC15] where it considers the results diversification problem in a publish-subscribe system for a large number of queries. They consider the Max-sum objective function that includes the text relevance, document recency, and results diversity. They continuously maintain a diversified top-k list for every query in the system. The core idea is the following: given a new document d_n and a query q, if when replacing the earliest document d_e with d_n in the top-k list of q, the diversity and relevance score increase, then the new document permanently replaces d_e . To efficiently process a large number of queries in the system, they propose a mechanism of grouping queries on blocks and a block-oriented query-processing. They use block-based query inverted lists and propose an effective block filtering technique using a document-at-a-time DAAT strategy. The proposed filtering technique is defined as follows: given a new document d_n , a block of queries b and the earliest document d_e in the results of queries in the block b, the documents d_n is filtered out by the block b if the upper bound of the diversity and relevance score of d_n considering the block b is below the lower bound of the diversity and relevance score of d_e considering b. They propose many of estimations to compute efficiently the upper bound and the lower bound for a block of queries.

However, their approach is limited to a specific model to compute the text relevance between a document and a query, the language model. This model is necessary in the proposed filtering technique in order to establish the upper bound over the text relevance. It is not clear how the text relevance score could be extended to other models such as the most popular ones, tf-idf with cosine similarity, BM25, etc.

Our approach is complementary to that of [CC15]. We adopt a flexible and easily generalizable method for indexing user queries (profiles) with sorted lists over the scoring and diversity criteria. A prof of this flexibility is the fact that we successfully

extended the index structure and algorithms from a relevance-only top-k processing to include diversity. The experimental results validate the efficiency and effectiveness of our method.

3 Continuous top-k queries in social networks

In this chapter, We present an efficient processing model for the continuous computation of the top-k query results for all the users of a social network. In social networks, users are interested in the streams of messages, and in particular, users want to be updated with the most relevant message. We focus on filtering the streams of messages using *the continuous top-k model*. *The continuous top-k model* continuously maintains the k messages highly ranked. We aim to develop a scalable system that be able to efficiently evaluate the continuous top-k queries using the *continuous approach* with a ranking function including social network criteria. We introduce our ranking model that includes the content-based, user-based, interaction-based and time-based components. While such ranking model is widely studied in recommendation system and Web search to improve ranking performance and search results, we are the first to propose such function in information stream systems.

In the social network, many events may procduce changes in the top-k messages. In this chapter, we present how to process the first categorie *continuously handled events* that includes the new message and the interaction with an existing message. As mentioned above, these events have a great impact on the top-k results, so we treat them on the spot.

We present our SANTA algorithm for continuous top-k computation which can efficiently handle our rich scoring function. SANTA process both message publications and user actions on existing messages. It indexes the top-k queries in a simple data structure based on the traditional IR inverted index and applies threshold-based techniques to prune the search space to enable the efficient processing. Existing algorithms for continuous top-k processing can be hardly extended with social network criteria in the scoring function and face the problem of heavy index updates when the top-k scores change, aggravated when the new, social network dimensions are added. Unlike them, SANTA minimizes the index updates by isolating them within a single dimension and can easily take advantage of its simple structure to extend to new dimensions. We illustrate this difference in this chapter, by comparing SANTA with an extension of the most popular state-of-the-art algorithm COL-Filter [HMA10]. We also present SANTA+, a variant of the SANTA algorithm that improves action processing. In fact, SANTA implies message re-processing and index re-traversal when processing a new action and this is time-consuming.

The rest of this chapter is structured as follows: section 4.1 presents the platform of information stream social network (ISSN) and introduces our proposed ranking function. In addition, we state the problem of computing the lists of best k messages for each user, and present our model for continuous top-k processing. section 3.2 introduces our algorithm SANTA and its varainat SANTA+ for improving action processing. section 4.3 presents the experimental evaluation of our algorithms. Finally, section 4.4 concludes this chapter.

3.1 Data and processing models

3.1.1 Data model

We consider information streams produced by the users of a *social network*. Streams are composed of *messages* (items), each message being characterized by a *content descriptor* that allows evaluating content similarity. We focus here on text-only messages, where content similarity is evaluated through vector models like tf-idf, and content descriptors may be represented as a vector of terms with a tf-idf weight associated to each term.

We model the social network as a pub-sub environment, where users publish messages and subscribe to information streams produced by other users in the network. The subscription queries are *implicit*, based on the *user profile*. A profile expresses the elements of interest for the user in messages and is also represented as a content descriptor, e.g. a vector of terms with their weights. Therefore, the importance of the content of a message m for a user u can be computed as the similarity between the content descriptors of m and of u's profile.

Users can also interact with messages, e.g. through likes, comments, forwarding, tagging as favorite, etc. We call *user actions* such interaction events; each message has a (possibly empty) set of associated user actions.

We consider social networks with asymmetric directed relations between users (such as for Twitter), which also cover the case of symmetric social networks (such as Facebook) by representing a two-way relation by two directed ones.

Definition 1. *An information stream social network (ISSN) $\mathcal{S}$ is a tuple $\mathcal{S} = (U, R, p, sim, f, s)$, where:*

- *U is a set of users.*
- *R=$\{(u_1, u_2)|u_1, u_2 \in U, u_1 \neq u_2\}$ is a set of non-symmetric relations between users; $(u_1, u_2) \in R$ means that u_1 "follows" the messages published by u_2.*
- *$p : U \rightarrow \mathcal{D}$ is a function associating a* profile *to each user. User profiles and message contents are both modeled as* content descriptors *in $\mathcal{D}$.*
- *$sim : \mathcal{D}^2 \rightarrow [0, 1]$ measures the similarity between two content descriptors.*

- $f : U^2 \rightarrow [0,1]$ *is a function associating to each couple of users* (u_1, u_2) *the importance of* u_2 *for* u_1 *in the social network.*
- $s : U \rightarrow \mathcal{I}$ *is a function associating to each user the information stream generated by that user.*

For text messages, with tf-idf based cosine similarity, a content descriptor $d \in \mathcal{D}$ is a vector of weights $d = [w_t | t \in \mathcal{T}]$, where $\mathcal{T}$ is a fixed dictionary of terms appearing in messages and $w_t \in \mathbb{R}^+$ is the weight of term t, with $w_t = 0$ for t not appearing in the message. By considering normalized weights, the cosine similarity function becomes $sim(d_1, d_2) = \sum_{t \in \mathcal{T}} w_{1t} w_{2t}$.

Note that the user relative importance function f is defined for any couple of users in the network, not only for those directly related through R. Like R, f is asymmetric. Depending on the design choices, the values of $f(u_1, u_2)$ may depend on many factors, of u_1 on the messages of u_2, the similarity between the content descriptors of u_1's profile and of u_2's profile, etc. and may change in time. In practice, each user has only a limited number of users of interest (with $f>0$), which results into reasonable effort to manage this information. In our case, we consider the influence of a limited neighborhood in the graph and of users interacting with the messages.

Definition 2. *An information stream* $I \in \mathcal{I}$ *is a couple* $I = (M, A)$*, where:*

- $M = \{(ts, d) | ts \in TS, d \in \mathcal{D}\}$ *is a set of messages, where* ts *is the timestamp of the message and* d *is the content descriptor of the message.*
- $A = \{(ts, u, m) | ts \in TS, u \in U, m \in M\}$ *is a set of user actions (e.g. likes, shares, etc.) on the stream messages.* ts *is the action's timestamp,* u *the user that realized it,* m *the target message of the action.*

Note that even if user actions may be of several types, we only focus here on actions as a proof of the interest of users for messages.

3.1.2 Scoring function

We consider here a scoring function $score(m, u)$ that expresses the importance of a message m for a user u, by combining content-based and social network factors. For simplicity, we consider here a linear combination of factors, but any monotonic function is compatible with our algorithm.

$$\begin{aligned} score(m,u) &= \alpha\ sim(m, p(u)) + (1-\alpha)\ social(m,u) \\ social(m,u) &= \beta\ global(m) + (1-\beta)\ f(u, u^m) \\ global(m) &= \gamma\ UI(u^m) + (1-\gamma)\ AI(m) \end{aligned} \tag{3.1}$$

Parameters $\alpha, \beta, \gamma \in [0,1]$ express the relative importance of the scoring function components. α expresses the balance between content-based similarity $sim(m, p(u))$

and social network based criteria. Inside $social(m, u)$, β gives the balance between global, user-independent factors ($UI(u^m)$, $AI(m)$) and user-dependent ones, expressed here by $f(u, u^m)$, the importance of the message emitter u^m for the user in the social network. Finally γ measures the balance between $UI(u^m) \in [0, 1]$, the global importance of the emitter u^m in the network, and $AI(m) \in [0, 1]$, the importance of the message given by the reactions it provoked, i.e. the actions realized on the message. We consider that new actions increase the value of $AI(m)$, i.e. $AI(m)$ is monotonically increasing with the number of actions on message m.

We also explore the introduction of a time dependent factor, expressing the loss of importance of messages in time. In fact, most applications consider recent messages are more important than older ones. As mentioned in section 2.1.7.1 two models were proposed to express the the freshness of information, sliding windows and time decay.

We consider a *decay function* [VAC12][SGFJ13], $TD : \mathbb{R}_+ \to [0, 1]$, monotonically decreasing and with $TD(0) = 1$. For a message m published at time t^m, the variation in time of the importance of message m for user u is expressed by the time-dependent scoring function $tscore : \mathcal{M} \times U \times TS \to \mathbb{R}_+$ such that for any moment $t \geq t^m$:

$$tscore(m, u, t) = score(m, u) \cdot TD(t - t^m) \tag{3.2}$$

Here $score(m, u)$ is the scoring function from (Equation 3.1) and expresses the initial importance of message m for user u at moment t^m.

Generally, only *order-preserving decay functions* are considered, i.e. functions that preserve in time the relative order of message scores. But even if this simplifies the continuous processing of top-k queries by preventing message reordering because of decay, maintaining time-dependent scores is unfeasible in practice.

Instead, we adopt the dual approach of *time-bonus functions* inspired from [CSSX09]. The idea is to give a score bonus to newer messages, instead of degrading scores in time. This produces the same effect as decay (penalizing older messages), with the advantage of fixed scores and of relative order preservation. A time bonus function $TB : \mathbb{R}_+ \to [1, \infty)$ is monotonically increasing and has $TB(0) = 1$. Given a fixed origin moment $t_o \in TS$, the time-dependent scoring function becomes time-independent:

$$tscore(m, u, t) = score(m, u) \cdot TB(t^m - t_o) \tag{3.3}$$

3.1.3 Problem statement

Given an ISSN and a scoring function such as (Equation 3.1) *or* (Equation 3.3)*, design an algorithm that efficiently computes and maintains the lists of best k mes-*

sages for each user, as new messages are published and new actions on the existing messages are registered.

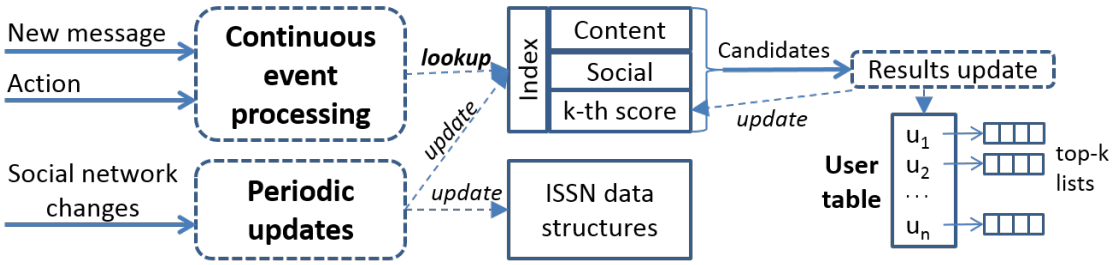


Figure 3.1: Model of continuous top-k processing in an ISSN

3.1.4 Processing model

We consider the ISSN essentially as an event-based system in which continuous top-k processing is realized through event handling. We focus here on *two main event types* that impact top-k results: *the publishing of new messages* and *user actions on the messages*.

Other events produce changes in the social network (e.g. new edge, new user, profile changes) and consequently impact the scoring parameters. Such events may have both *a local impact* on some users and *a global impact* on the ISSN, e.g. adding an edge from u_1 to u_2 locally impacts u_1 and u_2, but may also slightly change values for f in the ISSN. If the local impact may need continuous processing, the small global impact can be handled through *periodic updates* of the ISSN. Such changes are considered in the following *time consistency setting*: *the scoring parameters for a message are those at publishing/action time, changes to the ISSN do not modify the score of previous messages.*

In this work we focus on the continuous processing of the main events only. The study of handling the local impact of social network changes is presented, but only a brief discussion in the next section.

Figure 3.1 presents our model for continuous top-k processing. New message publishing and actions on messages are continuously processed. They provoke a lookup in the index structures, composed of a content-based index, a social index and a k-th score index. The result of this lookup is a set of candidate users for the top-k update. The role of the index is to drop from this set as many users not impacted by the event as possible, in order to enable efficient top-k processing. The update of the top-k lists provokes in return an update of the k-th score index. Social network changes are handled through periodic updates of the ISSN parameters, producing changes in the data and index structures. For simplicity, the local impact of these events is not represented here.

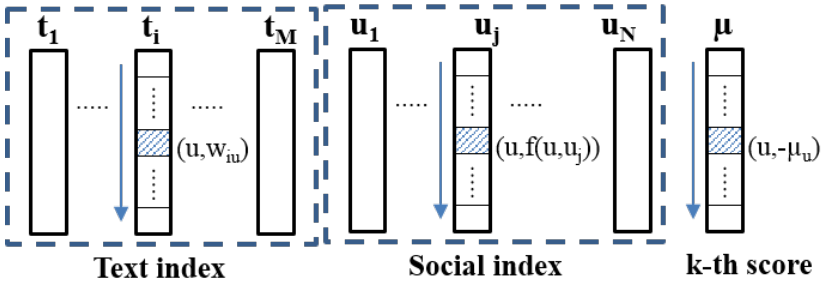


Figure 3.2: The SANTA index structure

3.2 The SANTA algorithm

The Social and Action Network Threshold Algorithm (SANTA) provides efficient continuous top-k processing based on a simple index structure, composed of sorted lists, traversed with threshold-based techniques to prune the search space.

Existing algorithms for continuous top-k processing can be hardly extended with social network criteria in the scoring function and face the problem of heavy index updates when the top-k scores change. Unlike them, SANTA minimizes index updates by isolating changes into a single dimension, while its simple index structure facilitates the extension to new social network dimensions. We illustrate this difference here, by comparing SANTA with an extension of the closest approach in the state of the art, the COL-Filter algorithm [HMA10].

3.2.1 Index and other data structures.

The SANTA index structure (Figure 3.2) is composed of a text index, a social index and the list μ of the current k-th score for each user. The text index is composed of lists for each term t_i, containing the users u that have t_i in the profile, sorted in descending order of the term's weight w_{iu}. The social index is composed of lists for each user u_j, containing users u for which u_j is important ($f(u, u_j)$>0), sorted by decreasing $f(u, u_j)$. The μ list is sorted in descending order of -μ_u (i.e. increasing μ_u). Note that μ is the only part of the index that needs updates during continuous top-k processing.

SANTA also manages a *user table* (Figure 3.1) to keep information about each user in the social network. The entry for user u in the table contains:

- the current top-k list for u;
- u's profile, as a list of (term, weight) couples;
- the list of users u' of interest for u, with the value $f(u, u') > 0$ for each one;
- the user importance $UI(u)$ in the social network.

The first component contains the current query results, while the other ones are necessary to evaluate $score(m, u)$ for any given message m.

Note an important scalability issue: SANTA processes messages on the spot and does not store them in the system. To get a previous message addressed by a new user action, we consider that the action event also provides the target message, which is the case in practice. Note also that in practice each user has only a limited number of users of interest (with $f(u, u') > 0$), requiring reasonable memory space for these lists and for the social index.

3.2.2 Scoring function

We consider the case of a scoring function such as (Equation 3.1) or (Equation 3.3), with cosine similarity for the textual content, but *any content similarity function monotonic in the index dimensions is compatible with SANTA*. For simplicity, let us consider first the time-independent case. Scoring function (Equation 3.1) can be written:

$$score(m, u) = a \sum_{t_i \in m} w_{im} w_{iu} + b\, f(u, u^m) + c\, G(m) \tag{3.4}$$

Here $G(m) = global(m) = \gamma UI(u^m) + (1-\gamma)AI(m)$ is the global, user-independent part of the score, $a = \alpha$, $b = (1-\alpha)(1-\beta)$ and $c = (1-\alpha)\beta$.

If we note $F(m, u) = score(m, u) - \mu_u$, a message m will enter the top-k of u iff $F(m, u) > 0$. Given the form of $score(m, u)$ in (Equation 3.4), it is easy to remark that for a given m, $F(m, u)$ is a constant ($c\, G(m)$) plus a positive weighted sum in the index dimensions w_{iu}, $f(u, u^m)$ and $-\mu_u$. Consequently, F is monotonic in the index dimensions, which allows threshold strategies such as TA [Fag02] to traverse the index lists in order to get candidates u for top-k change.

In the case of the time-dependent scoring function (Equation 3.3), we have $F(m, u) = score(m, u)\, TB(t^m - t_o) - \mu_u$, i.e. all the components of the score are multiplied by the same positive factor. This does not change the monotony of F and the same algorithm as for time-independent scores can be applied.

3.2.3 The algorithm

Algorithm 3.1 presents the SANTA algorithm, as a set of two event handlers, *newMessage* and *newAction*. Both use the same approach, expressed by the *getCandAndUpdate* method: for each candidate user extracted from the index, check if the message enters the top-k for that user; if so, update its top-k and the corresponding entry of μ. The difference is that *newMessage* already has the incoming message, while

Algorithm 3.1 The SANTA algorithm

Input: message m, action a, index I, user table U
newMessage (m, I, U)
getCandAndUpdate(m, I, U)
end
newAction (a, I, U)
$m \leftarrow$ *getMessage*(a)
getCandAndUpdate(m, I, U)
end
getCandAndUpdate (m, I, U)
foreach c **in** *getCandidates*(I,m) **do**
$ue \leftarrow$ *getUserEntry*(U,$c.user$)
if $c.upperBound > ue.kthScore$ **then**
$s \leftarrow$ *computeScore*(m,ue) *//compute real score*
if $s > ue.kthScore$ **then** update ue and $I.\mu$
end if
end foreach
end
getCandidates (I, m)
initTraversal(I,m); $result \leftarrow \emptyset$; $threshold \leftarrow \overline{F}(m)$
while $threshold > 0$ **do**
$u \leftarrow$ *nextIndexUser*(I)
$result \leftarrow result \cup (u, \overline{score}(m))$
$threshold \leftarrow \overline{F}(m)$
end while
return *result*
end

newAction retrieves it from the action. For *newAction*, the new action will increase the value of $AI(m)$, so $G(m)$ grows and the $F(m,u) > 0$ condition will produce more candidates from the index.

Note also a subtle difference between *newAction* and *newMessage*. Since processed messages are not stored in the system, retrieving the message from the actions means a new processing of the message to extract terms and their weights. This may add a quite significant extra processing time for actions.

getCandAndUpdate checks each candidate c returned by the index traversal (*getCandidates*), where c contains both the user $c.user$ and an upper bound estimation $c.upperBound$ for $score(m, c.user)$. The entry ue of $c.user$ in the user table is necessary to compute the real $score(m, c.user)$ with *computeScore*(m,ue) and to get the k-th score of $c.user$. To avoid systematic computation of the real score (costly operation), $c.upperBound$ is first checked against the k-th score; the computation is not necessary if $c.upperBound$ is not greater. If the real score s exceeds the k-th score, then m enters the top-k of $c.user$. Both the entries for $c.user$ in the user table (for the top-k list) and in the μ list are updated; $c.user$ goes downward in μ since its k-th score increases.

The threshold strategy for limiting the number of candidates is implemented by the *getCandidates* method. For message m, *initTraversal* selects the related lists from the index (those for the terms contained in the message m) and computes the coefficients of $F(m,u)$. The index lists traversal may follow any threshold algorithm strategy through the call to *nextIndexUser*, which returns the next user in some of the lists. The best known one is the TA strategy [Fag02], which considers lists in a round-robin order, but other strategies are possible. We define $\overline{F}(m)$ as being $F(m,u)$ applied to the last visited value in each index list (or to its maximum value if not yet accessed). Since $F(m,u)$ is monotonic and index lists are traversed in descending order of scores, $\overline{F}(m)$ gives the threshold (decreasing during index lists traversal) that $F(m,u)$ cannot exceed for any new candidate u to be found in the index.

Any new candidate found while $threshold > 0$ may have m in its top-k; it is added to the list together with its upper bound score $\overline{score}(m)$, computed like $\overline{F}(m)$ but excluding the μ list. The traversal stops when $threshold \leq 0$.

Figure 3.3 illustrates an example of execution of *getCandidates* with a TA strategy. We consider a new message m, published by user u_1, containing two terms, t_1 of weight 0.6 and t_2 of weight 0.4. Hence, only lists for t_1 and t_2 in the text index, for u_1 in the social index and μ are concerned. We consider a scoring function with a=0.5, b=0.3, c=0.2 and $G(m)$=0.1. The TA strategy considers candidates and computes the threshold line-by-line; for the first line candidates are u_8, u_3, u_4 and u_2, and threshold $\overline{F}(m)$=a·(0.6·*0.5*+0.4·*0.4*)+b·*0.4*+c·0.1-*0.25* = 0.12 > 0. The four candidates are added to the result list, with an upper bound $\overline{score}(m)$=0.12+0.25=0.37. For the next line in the index lists, $\overline{F}(m)$=a·(0.6·*0.4*+0.4·*0.3*)+b·*0.3*+c·0.1-*0.3* = -0.01 < 0. The traversal stops since for all the other u in the index $F(m,u)$<$\overline{F}(m)$<0;

only the previous four candidates are returned. Consider now the processing of an action on m. For simplicity, we consider in this example the same values for μ, even if they changed since the arrival of m. Since $AI(m)$ increases, $G(m)$ also, and $score(m, u)$ augments for any u. If, e.g. now $G(m)$=0.2, the index traversal will find threshold values that increase with $c{\cdot}\Delta G(m)$=0.02 for every line. The traversal will accept also candidates from the second line (u_{11}, u_5, u_6 and u_7), since now $\overline{F}(m)$=-0.01+0.02>0. Their upper bound is $\overline{score}(m)$=0.01+0.3=0.31. For the third line however, $\overline{F}(m)=a{\cdot}(0.6{\cdot}0.3+0.4{\cdot}0.3) + b{\cdot}0.1+c{\cdot}0.2-0.31$ = -0.09 < 0.

3.2.4 SANTA+: improving action processing

Algorithm 3.2 The SANTA+ algorithm

Input: message m, action a, index I, user table U, window W

```
newMessage (m, I, U, W)
  me ← storeMessage(m,W)
  initPos ← initTraversalPos(m,I)
  getCandAndUpdatePos(initPos, me, I, U, W)
end
newAction (a, I, U, W)
  me ← getMessageEntry(a, W)
  if me exists then //use the stored message entry
    increase ← updateDelta(me, a)
    foreach ce in me.candidates do
      ue ← getUserEntry(U,ce.user)
      ce.score ← ce.score + increase
      if ce.score > ue.kthScore then update ue and I.μ
      elsif me.delta ≤ ue.kthScore − ce.score then
        remove ce from me.candidates
      end if
    end foreach
    getCandAndUpdatePos(me.indexPos,me,I,U,W)
  else //use the SANTA algorithm
    getCandAndUpdate(getMessage(a), I, U)
  end if
end
getCandAndUpdatePos (pos, me, I, U, W)
  (newPos,cand) ← getCandidatesPos(I,me.msg,pos)
  foreach c in cand do
    ue ← getUserEntry(U,c.user)
    s ← computeScore(me.msg, ue) //compute real score
    if s > ue.kthScore then update ue and I.μ
    if me.delta > ue.kthScore − s then add(me,c.user,s)
  end foreach
  me.indexPos ← newPos
end
```

Action handling with SANTA implies message re-processing and index re-traversal. We propose an improvement with the SANTA+ variant, which *stores processed messages* and *keeps for each of them the list of candidates (with the real score)* found at message publishing, that may be interested by the message if an action increases its score.

When an action occurs, the stored candidates are first checked; this is fast, since their real scores are already computed. Then the index traversal can continue to discover new candidates, but *starting from the previous position*, not from the beginning. Since we do not want to store all the messages, we consider *a fixed size message window*. Most actions are close in time to the message publication, so the message has great chances to be in the window when the action occurs. If not, the action is processed with the basic SANTA algorithm.

Each message m in the window keeps the following information:

- content descriptor of the message;
- previous position in the index, after last action or after message arrival; since the μ list is dynamic, the position in μ is kept as the last read μ value;
- number of actions on m and maximum increase of the AI score $\Delta AI(m)$;
- set of candidates represented as (user, score) couples.

As mentioned in subsection 3.1.2, $AI(m) \in [0, 1]$ is monotonically increasing with the number of actions on m. If $AI(m)_{max}$ is the upper bound of $AI(m)$, then $\Delta AI(m) = AI(m)_{max} - AI(m)$ is the (decreasing) maximum bonus m can get with new actions.

Algorithm 3.2 also presents the SANTA+ algorithm. Unlike SANTA, here the index traversal and candidate processing are realized by *getCandAndUpdatePos* from a given position in the index. The *newMessage* handler stores the message in the message window, gets the initial index position with *initTraversalPos* and handles candidates through *getCandAndUpdatePos*. Two main differences distinguish *getCandAndUpdatePos* from SANTA's *getCandAndUpdate*. First, we manage the index position: *getCandidatesPos* traverses the index like SANTA, but from a given starting position and get candidates together with the new index position. Also, the final position is stored in the message entry *me*. Next, candidates are also inserted into the message's candidate list if they have chances to have m in their top-k. Note that here the real score is always computed, because needed for the message's candidate list.

The *newAction* handler distinguishes two cases. If the message is still in the window, *updateDelta* computes the score increase given by the action and decreases $\Delta AI(m)$. Then each candidate in m's list augments its score and is tested for top-k. Also, if the candidate has no chances to enter the top-k (because the current k-th score is high), it is removed from the list. Finally, new candidates are extracted from the index, by continuing the traversal from the stored position, by using *getCandAndUpdatePos*.

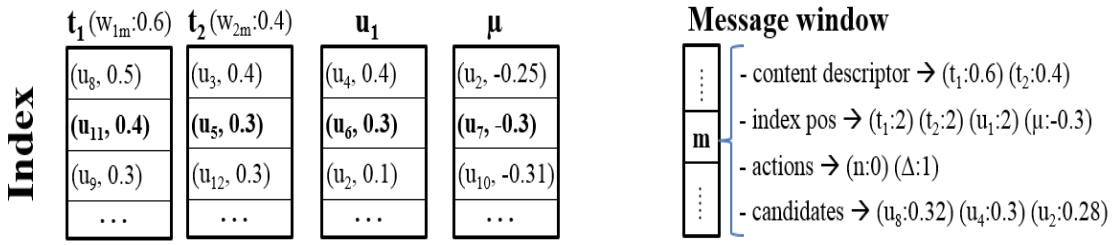


Figure 3.3: Execution example for SANTA and SANTA+

In the case where the message has exited the window, it uses the SANTA algorithm through *getCandAndUpdate*.

Figure 3.3 also illustrates SANTA+ execution. When m arrives, it enters the message window and the index traversal returns the same four candidates. In the example only three of them enter the candidate list of m, but not u_3, e.g. because $score(m, u_3)$ is too low compared to μ_{u3} and actions cannot compensate the difference. When the action on m occurs, the message entry is updated with incremented n and decreased $\Delta AI(m)$. Then m's candidates (u_8, u_4, u_2) are checked for top-k considering the score increase produced by the action. Some of them may exit the list if their decreased $\Delta AI(m)$ is not enough to reach the current (increased) top-k. Then, index traversal is continued from the last position (line 2 for t_1, t_2, u_1 and μ=-0.3) and returns new candidates (u_{11}, u_5, u_6, u_7, as for SANTA) that are checked for top-k. The message entry is then updated with the new index position and the new index candidates with chances to get m in their top-k.

Remarks.

- The SANTA index allows a simple and efficient parallelization of the SANTA and SANTA+ algorithms. By partitioning the set of users on N machines, each one can build its own index and user table on that subset of users. Each incoming message or action is processed in parallel by all the machines, on their local index and/or message window, with no dependencies between them. Results are distributed in the various user tables on the N machines.
- With the time consistency setting adopted in our processing model (subsection 3.1.4), the SANTA algorithm is not impacted by periodic changes of the ISSN, since the index is traversed from the beginning for each event. For SANTA+, a periodic change requires emptying the message window to ensure consis- tency. The local impact of ISSN changes is also easy to handle, e.g. a new user u requires a new entry in the user table and the insertion of u in the text index given the profile terms, a new edge (u1 ,u2) requires the insertion of u1 into the social index of u2 with some default importance, etc.

Theorem 1. *The SANTA and SANTA+ algorithm are correct and complete.*

Proof. Both algorithms update the top-k lists after evaluating candidate scores. To be correct and complete, each algorithm must (i) not miss a candidate that would update its top-k, and (ii) must use the right score for each candidate. For SANTA, the monotonicity properties of the threshold algorithm ensure that a candidate c not delivered by the index respects the condition $F(m, c.user) \leq \overline{F}(m) \leq 0$ when *getCandidates*() stops. Since $F(m, c.user) \leq 0$, m cannot enter the top-k of $c.user$. The same properties ensure that for any candidate c returned by the index, $score(m, c.user) \leq \overline{score}(m) = c.upperBound$. A candidate returned by the index is not evaluated if $c.upperBound \leq ue.kthScore$, but in this case we have $score(m, c.user) \leq c.upperBound \leq ue.kthScore$, so m cannot enter the top-k of $c.user$. This guarantees that condition (i) is satisfied. Condition (ii) is also true, because for every candidate that updates its top-k, the score is directly evaluated on the user entry. Both *newMessage*() and *newAction*() use the same algorithm, so SANTA is correct and complete. SANTA+ uses the same algorithm as SANTA for *newMessage*(), excepting the upper bound test, which results in testing all the candidates returned by the index, so (i) and (ii) are satisfied. For *newAction*(), the same is true if the message is not in the window, since we use SANTA. Let us consider now the case when the message is in the window. We first check that the candidate scores stored in the message window are always correct. These scores are correctly set by *newMessage*() and updated by *newAction*() for any action on that message. There is no other event that may change the score, which means that scores are always up to date and correct. Top-k updates are based either on these scores, or on the user entry, so they are always correct, i.e. (ii) is satisfied. For condition (i), let us consider a "good" candidate missed by the algorithm. We saw above that candidates not returned by the index cannot be "good" candidates. So this may happen only if the candidate did not enter the window list or has been removed from. But this happens only if $me.delta \leq ue.kthScore - score$, i.e. the candidate has no chance to get the message into its top-k in the future. Hence, no "good" candidate is missed, so condition (i) is satisfied. □

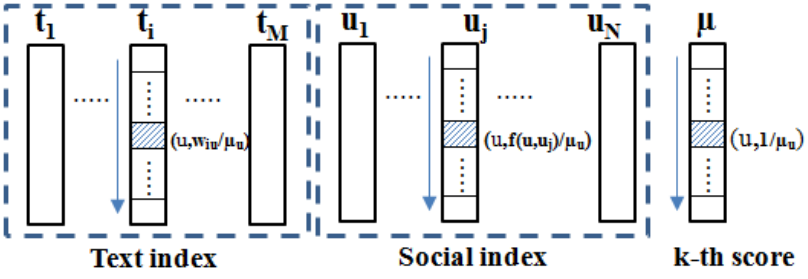


Figure 3.4: The CF+ index structure

3.2.5 CF+: an extended version of COL-Filter

As mentioned above, the closest approach to ours is the COL-Filter algorithm [HMA10], that also uses an index based on sorted lists, but for scoring functions limited to textual similarity. The COL-Filter index is similar to the SANTA text index, with the difference that μ is incorporated into the index by dividing each w_{iu} score by μ_u. Similarly to the SANTA condition for entering the top-k ($F(m,u) = score(m,u) - \mu_u > 0$), the condition for COL-Filter is $F'(m,u) = score(m,u)/\mu_u > 1$. This strategy reduces the number of dimensions to accelerate index traversal, but extends the need for updates to all the dimensions.

To compare SANTA and COL-Filter strategies, we propose CF+, an extension of COL-Filter to our scoring function, as follows. In CF+, the condition to enter the top-k becomes $F'(m,u) = a \sum_{t_i \in m} w_{im} w_{iu}/\mu_u + b\ f(u,u_m)/\mu_u + c\ G(m)/\mu_u > 1$. Since F' is a positive weighted sum of w_{iu}/μ_u, $f(u,u_m)/\mu_u$ and $1/\mu_u$, CF+ can use an index structure (Figure 3.4) very similar to SANTA: textual index lists for each term t_i with w_{iu}/μ_u values (like COL-Filter), social index lists for each user u_m with values $f(u,u_m)/\mu_u$ and a k-th score list with the values of $1/\mu_u$. Based on this index, CF+ handles messages and actions exactly like SANTA, and uses the same threshold strategy for index traversal, with the specific difference of the stop condition: $F'(m,u) \leq 1$.

In the next section we experimentally compare CF+ with SANTA variants and show that the number of updates required by the COL-Filter strategy is prohibitive when adding social network criteria.

3.3 Experimental evaluation

Dataset and scoring function

The graph. The ISSN used in the experiments is a subgraph extracted from Twitter. It contains almost $|U|$=104 000 users with around $|R|$=18 million direct links between them. The community was built starting from around 200 accounts of known French politicians and journalists, by adding part of their followers, more precisely those having a number of followees within the community, above a given threshold. This method resulted into a coherent social network, with a good density of links.

Messages and actions. For each user the last 200 tweets were extracted (or all, if less) with the corresponding user actions (retweet, reply and mark as favorite). The terms extracted from tweets are the hashtags, but also common nouns and proper nouns, using the TreeTagger[1] tool. We only kept non-empty messages (with at least one term) and their actions, which results in around 1.25 million messages

[1]http://www.cis.uni-muenchen.de/ schmid/tools/TreeTagger

and 180,000 actions. Each message contains only a few terms, between 1 and about 10 in our corpus, with an average between 3 and 4 terms.

Terms and user profiles. A dictionary of around 187,000 terms was built with message terms that are used by at least 5 users. For each user, the profile contains all the dictionary terms that occur in his messages. The profile size goes from 1 to around 1000 terms, with an average size of 125 terms. The weights of the profile terms, based on the *tf* and *idf* values, are computed by considering that the messages of each user form a single document.

Social relations. For the values of the $f(u_1, u_2)$ function in the ISSN, we combined only two factors: the existence of a direct link (u_1, u_2) and the number of actions made by u_1 in relation with u_2. Besides the 18 million direct links, we obtained almost 1 million extra non zero values for f. These 19 million relations of interest are retrieved in the 104,000 social index lists.

Scoring function. The scoring function uses as default coefficient values α=0.5 (equal weight for content and social criteria), β=0.25 (25% weight for the global, user-independent criteria and 75% for the local relations of interest expressed by the f function) and γ=0.4 (40% weight for the user importance UI and 60% for the action impact AI). For $UI(u^m)$ we use the Klout[2] score, which expresses the influence of users in the main social networks, normalized to the $[0, 1]$ interval. For $AI(m)$ we consider only the influence of the number n of actions on the message m: $AI(m) = 1 - e^{-\lambda_a n}$, with $\lambda_a = 0.5$.

For the time bonus we use a linear function $TB(t^m - t_o) = 1 + (t^m - t_o)/T_b$, where T_b is the period of time after which an extra bonus equal to the time-independent $score(m, u)$ is earned.

Experimental protocol. If not specified, the default values used in the experiments are: k=10, α=0.5, β=0.25, γ=0.4 and no time bonus. The message window size for SANTA+ is not bounded in the experiments, but, as illustrated below, the impact of a reasonably large bounded window is small, wi*Experimental protocol*th no consequence on the conclusions. For most experiments we use the whole dataset, with messages and actions processed in timestamp order.

Since we aim at measuring performance in a stable status, we consider an *initialization phase*, followed by *the measure phase*. The stream of messages and actions in split in two almost equal parts, initialization considers the first 600,000 messages and 90,000 actions, then measures are realized on the remaining 650,000 messages and around 90,000 actions. In experiments considering a subset of these messages and actions, we specify the balance between initialization and measure. Algorithms are programmed in Java and run on a multi-core server. Memory requirements are here of about 1 GB for the index and 0.5 GB for the user table.

[2]https://klout.com

Comparing CF+ and SANTA variants

We compare the CF+ and SANTA algorithms by measuring the average processing time for new messages and new actions. To illustrate the drawback of the COL-Filter approach, we measured separately the time needed for index search and for index/result updates. Besides SANTA and SANTA+, we considered a variant of SANTA called SANTA$_{CF}$, which handles index updates in the same way as CF+, in a separate phase, while SANTA and SANTA+ realize them during search. This allows measuring in a reliable way the update time for CF+ and SANTA$_{CF}$, while for SANTA and SANTA+ we only can measure the aggregate time.

Figure 3.5 presents this comparison. Measures for the average processing time per message show that CF+ is faster than SANTA$_{CF}$ for search (0.11 vs 0.3 ms), but much worse for update (5.09 vs 0.76 ms); globally SANTA$_{CF}$ is almost 5 times faster. The mix of search and update realized by SANTA is beneficial, the global time for SANTA being comparable with the search time only for SANTA$_{CF}$. SANTA+ needs slightly more processing time than SANTA because of the message window management.

Measures for action processing show that CF+ is not adapted for action handling, the update time (101 ms) is two orders of magnitude larger than for SANTA. We notice the effectiveness of SANTA+ for dealing with actions, materialized by an execution time almost 10 times faster than for SANTA.

In conclusion, SANTA algorithms provide an effective solution for message and action processing when the scoring function includes new, social network criteria. Their simple index structure favors fast index updates, which is not the case for the COL-Filter approach, which obtains strongly degraded update times. Note that POL-Filter, the improved variant of COL-Filter presented in [HMA10] only reduces the update time with about 11%, which does not change the conclusions of this comparison.

SANTA and SANTA+

We follow the comparison between SANTA and the SANTA+ variant with an in-depth analysis of their behavior. Besides the execution time, we measure the number of candidates whose score is computed and the number of top-k lists updates. To evaluate the impact of the time-dependent factor in the score, we consider two cases: without time bonus and with moderate time bonus, corresponding to T_b=15 days.

To compare with the time bonus case, we use here a dataset restricted to a period of about 10 months and better adapted to time-dependent score analysis. It contains about 500,000 messages and 75,000 actions, of which 300,000 messages and 40,000 actions are used in the initialization phase.

The measures, in the table on Figure 3.6, correspond to an average over the test dataset. *In the no time bonus case* measures are similar to those from the comparison

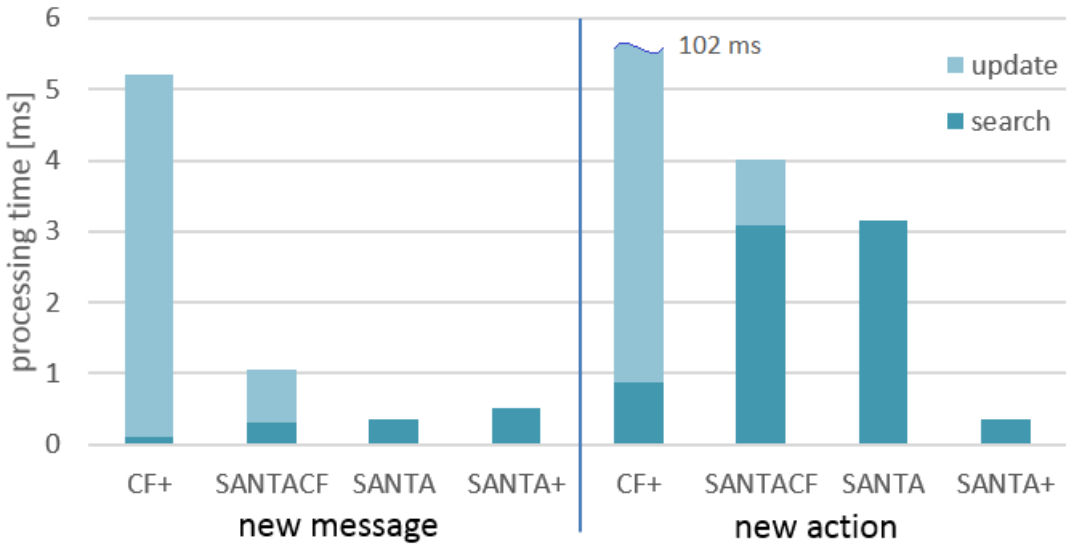


Figure 3.5: Comparison between CF+, SANTA and SANTA+

with CF+. For message processing, SANTA+ is slower than SANTA, which is explained by the message window management, but also by the higher number of candidate score evaluations. Indeed, SANTA avoids computing the real score for part of the candidates (only 250 up to 518 in average per message), which is not possible for SANTA+. In average, less than one top-k list is impacted for each message.

Since not all messages receive user actions, in parenthesis are reported measures for the subset of messages with actions, for a better comparison with the action processing case. All the values significantly increase in this case. This suggests that messages provoking actions are in a great measure published by influential users, which results into higher social and global scores, longer index traversals, more candidates and more chances to enter the top-k. In this light, the strong difference in processing time for messages and actions in SANTA (0.35 vs 3.15 ms) is highly reduced (2.21 vs 3.15 ms).

For actions, SANTA+ behaves much better than SANTA. We separately measured the time spent to search the message's candidate list in the message window and the retrieval of new candidates from the index. SANTA+ is up to an order of magnitude faster, by taking advantage of the preprocessed candidate list in the window, already pruned and with computed scores. This leaves only 210 candidates to evaluate, instead of 2100 for SANTA.

In the case of scoring with time bonus, new messages have more chances to enter the top-k and the action impact is amplified for recent messages. For message processing, this results in an increase of the number of candidates and thus of the execution time (with about 50% here). This also explains the strong increase of the ratio of updated top-k lists compared to the no bonus case.

			Without time bonus		With time bonus	
Measure			SANTA	SANTA+	SANTA	SANTA+
message	execution time [ms]		0.35 (2.21)	0.51 (2.59)	0.6	0.75
	# candidates		250 (1503)	518 (2519)	379	702
	# updated top-k		0.82 (5)		18	
action	execution time [ms]	window	3.15	0.06	7.01	0.89
		index		0.29		1.51
	# candidates	window	2100	109	4220	869
		index		210		1354
	# updated top-k		27		507	

Figure 3.6: Detailed comparison between SANTA and SANTA+

For actions, the impact is stronger, especially because of the increase of the number of candidates given by the index, both for SANTA and for the index-based part of SANTA+. The extent of this impact is also measured by the strong increase of the number of updated top-k lists.

Remark: the comparison for actions does not measure the time needed to re-process the message in SANTA. This time is difficult to evaluate and hardly depends on the access time to the message and on the processing method. In our context, for instance, only the average processing time for the TreeTagger tool is about 7 ms / message, but globally the extra-time may be much higher. In conclusion, SANTA+ is much better than SANTA for action handling, but needs a slightly longer time for message processing. In a context with many user actions, the choice

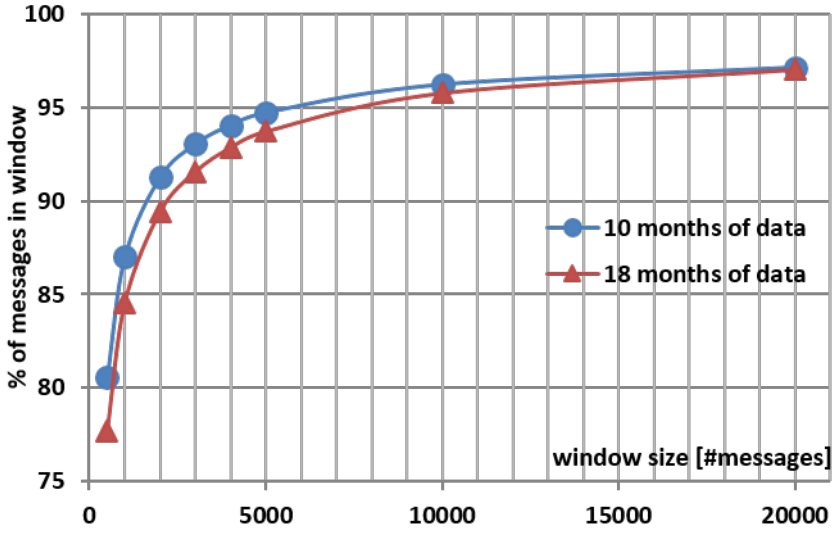


Figure 3.7: The impact of a bounded message window

of SANTA+ is natural, while SANTA is more appropriate in social networks with few user actions. Also *a dynamic combination* of SANTA and SANTA+ is possible, by applying SANTA+ only on "important" messages, with more chances to provoke interaction. The use of a time bonus increases the probability of top-k updates and the number of candidates, especially for action processing. A more detailed analysis of this impact is given below.

SANTA+ and the message window size

The previous comparison considered an unbounded message window for SANTA+, but for memory constraints, a fixed size window is necessary in practice. The impact on SANTA+ is that message lookups in the window upon a new action may fail if the message exited the window; in this case SANTA+ uses SANTA. We measure here the impact of the window size on SANTA+ as the success ratio of message lookups in the window. The execution time can then be estimated as a linear combination of those of SANTA+ and SANTA, with this success ratio.

Figure 3.7 presents the variation of the success ratio as a function of the window size. We use two datasets, one with messages over about 10 months (500,000 messages and 75,000 actions) and another one over about 18 months (1,100,000 messages and 170,000 actions). The variation follows the same shape in both cases, with about 80% of success for a window of 500 messages, 90% for a size of 2000 and 95% for a size of about 8000.

In conclusion, a small amount of memory for the message window (several MB here) is enough to keep the good performances of the SANTA+ algorithm.

Varying k

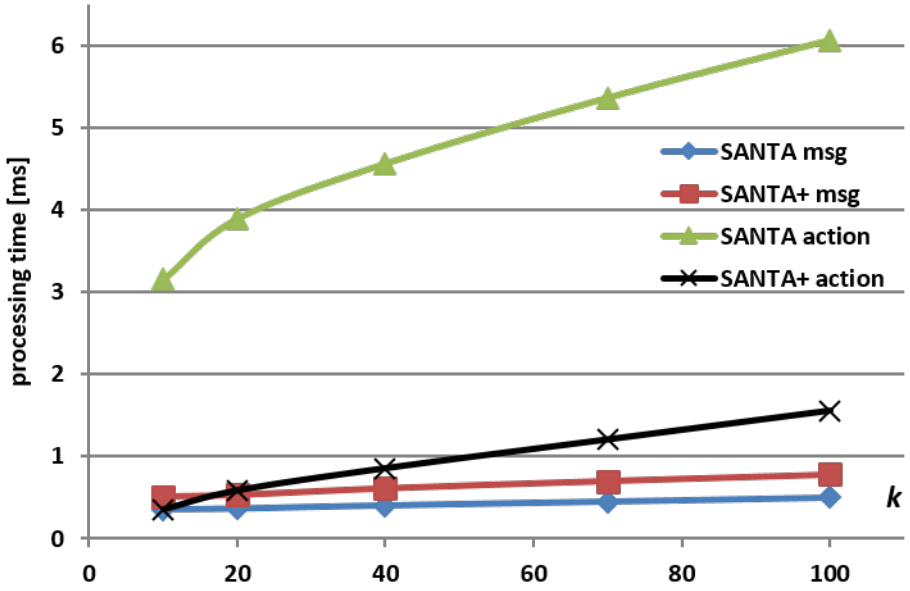


Figure 3.8: Variation with k

We measured the impact of k on the processing time for messages and actions, by varying k between 10 and 100. The results in Figure 3.8 indicate a small, quasi-linear increase of time with k for message processing for both SANTA and SANTA+. The increase rate is clearly larger for action processing for both algorithms, bur remains linear.

Time-dependent scoring

We noticed above the influence of a time bonus on the processing time, given the greater scores of newer messages. Figure 3.9 presents a finer analysis of this phenomenon for message processing and Figure 3.10 for action processing , when the bonus period T_b varies from 1 to 180 days. The horizontal axis is graduated in $1/T_b$ with T_b expressed in days, the 0 value corresponding to the no time bonus case, then points at T_b=180, 90, 45 and 15 days. To zoom on the most interesting values, since all the curves have the same shape, with an initial increase followed by a stationary zone, the values for T_b=1 day are not shown - they are slightly greater than for 15 days. For SANTA+ action processing we also separately show the time spent with candidates from the message window (SANTA+ window) and from the index (SANTA+ index).

In all the cases we observe a real impact of the time bonus on the processing time, even for the smaller values of the time bonus. The increase, larger for action process-

ing, remains limited to reasonable values. Only SANTA becomes rather expensive for action processing, but this only enforces the recommendation of using SANTA+ in this case.

Varying α, β, γ.

The behavior of our algorithms may depend on the relative weight given to the textual, graph, user importance or action components of the score. This balance is expressed by the α, β and γ parameters of the scoring function. We measured the processing time for various combinations of α, β and γ around the default values ($\alpha = 0.5, \beta = 0.25, \gamma = 0.4$), but not only.

As can be seen on the left side of Figure 3.11, for messages, variation α has a considerable impact on the processing time for both SANTA and SANTA+. Remember that α expresses a compromise parameter between textual (text index) and social network (social index and importance of the user) components. We can note that higher values of α results in more user candidates retrived from text index, which leads to a higher processing time. For actions, the right side of Figure 3.11 shows the processing time decreases slightly when α grows for both SANTA and SANTA+. This is obvious since a large value of α means less weight for the social network factors in which actions are included, therefore, new actions are less likely to update top-k.

Variation of β has a significant impact when increasing, illustrated on the left side of Figure 3.12. For messages, the processing time strongly decreases when β grows. This can be explained through the fact that big β values give a great weight to the user importance (the action impact is 0 for new messages), which increases the

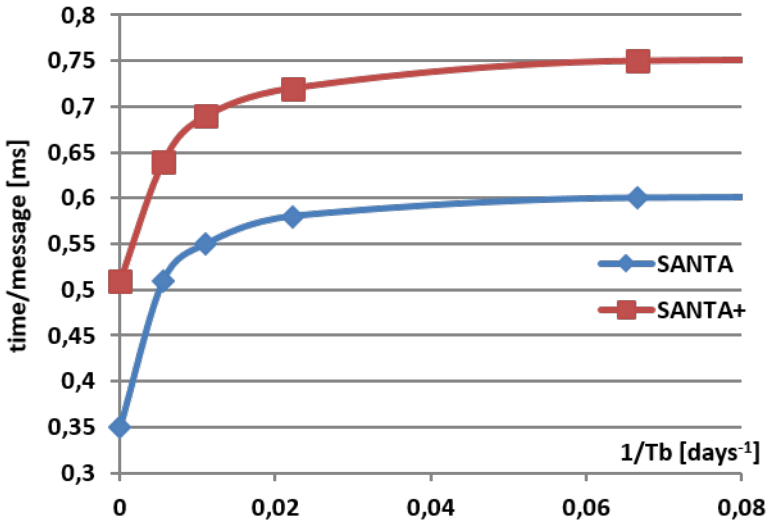


Figure 3.9: Varying the time bonus for message processing

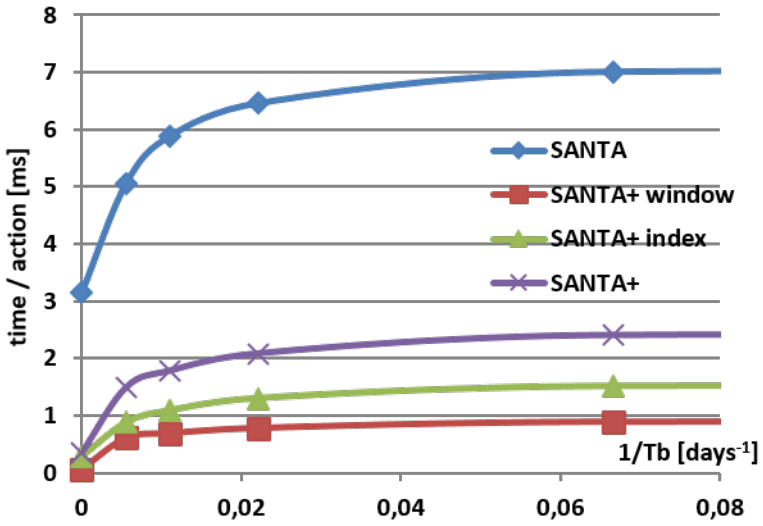


Figure 3.10: Varying the time bonus for action processing

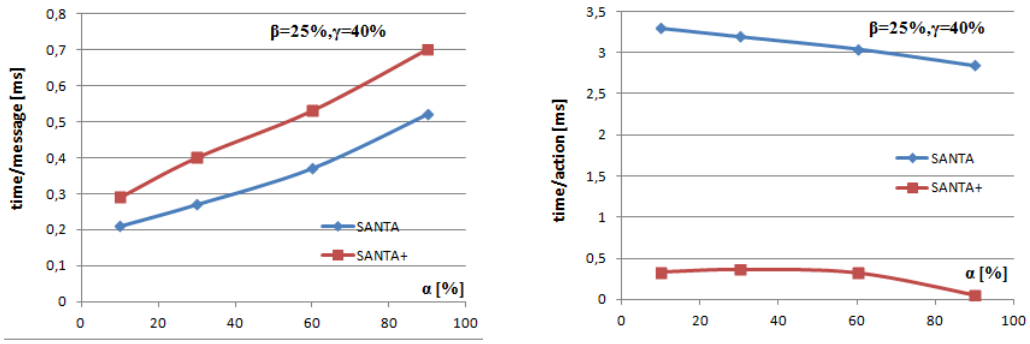


Figure 3.11: Varying α

impact of important users, whose messages will have high scores for all the users in the network. On one hand this will populate top-k list with these messages and decrease the chances of other user messages to enter the top-k, which results in few candidates from the index, few updates and fast execution time. For actions, SANTA has a less abrupt but similar variation, since it is based on the index but the score increase brought by the action attenuates the influence of the user importance. Unlike SANTA, SANTA+ is much less affected by the index and keeps a small increase of the processing time withβ.

Variation of γ can be seen in Figure 3.13, both SANTA and SANTA+ are stable, with only minor variations.

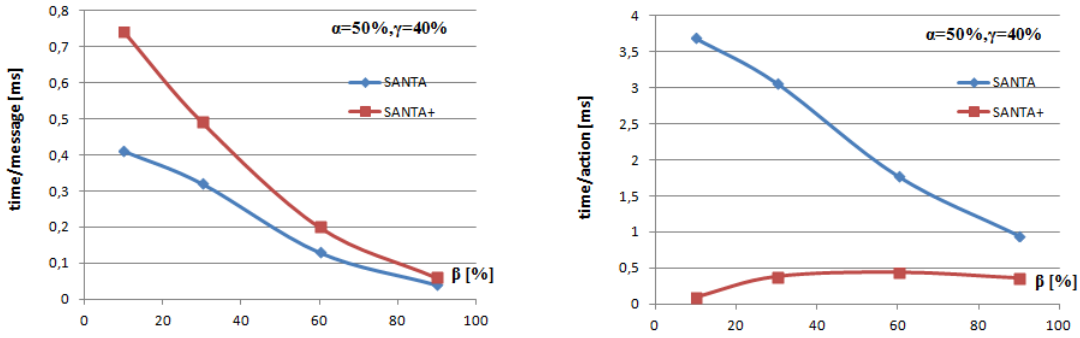


Figure 3.12: Varying β

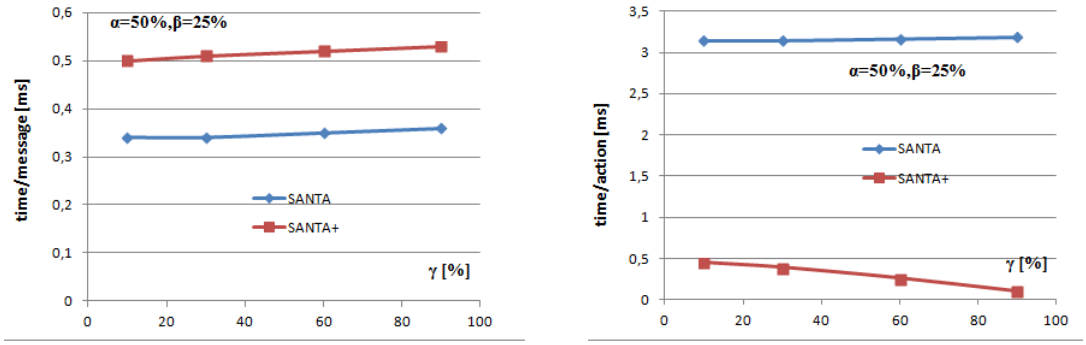


Figure 3.13: Varying γ

3.4 Summary

Prior works have proposed efficient algorithms for continuous top-k processing of information streams using query-based index data structures to prune the search space. However, these studies have considered simple scoring functions including textual and time factors; [VAC12] considers a global importance of message (query-independent component) which can be assimilated to our interaction-based score components. Moreover, these studies are not adapted to the social network context since the need of many index updates. As mentioned above, these studies include the value of the k-th score (having frequent changes) into each index dimension. This can be effective in case of short queries (a few terms). However, for social networking environments with implicit subscription queries based on user profiles (long queries), these indexes fail to effective processing since the number of updates grows with the size of the query and with the number of dimensions (that increases when introducing social network criteria).

We have presented the adopted ranking model that includes the content-based, social-based and time-based components. The social component in our ranking model includes the global importance of the message and the relationships between users. We have considered the global importance of the message is dynamic when

the message has interactions, that produces changes in the scores of messages in top-k sets. Next, we have proposed an index structure composed of the *text index*, *social index*, and *list* μ. This structure adapts to the social network context and optimizes index updates following the changes in scores of messages. In contrast to previous work, index updates in the proposed structure are not related to users queries (*text index*) or users "friends" (*social index*) and are occurred in a single dimension (*list* μ). We have introduced the algorithm SANTA and its variant SANTA+ for continuous top-k queries processing that can handle both new messages and new actions on the spot. We have compared SANTA with an extension of the closest approach in the state of the art, the COL-Filter algorithm. In conclusion, our system provides an effective solution for message and action processing when the scoring function includes new, social network criteria.

4 Diversity-aware continuous top-k queries in social networks

In this chapter, we tackle the problem of query results diversity. In the *continuous top-k model* presented in the previous chapter, users may receive similar and redundant messages in their top-k sets since the ranking model is based on relevance-only criteria. Result diversification aims at avoiding redundancy and too homogeneous results. In this chapter, we aim to continuously keep diversified top-k result lists over streams of messages.

The first challenge in this context is how to define and measure the diversity for query results. The *content diversity* of a results set is generally measured either by the average or the minimum of the distances between all the results. The general approach to add diversity to top-k querying is to use a *bi-objective scoring function* that combines relevance and diversity. In this chapter, we focus on the content-based diversification where we want to maximize the dis-similarity between messages in the top-k list. The seconde challenge is how to integrate diversity into the *continuous top-k model* while maintaining the efficiency of the system. The existing works on the continuous version of the diversification problem are not adapted to large pub/sub environments such as social networks, since they evaluate each new item with all the queries. The only work to date that proposes a diversity-aware method adapted to a large number of queries is [CC15]. In this context, this Chapter proposes a smooth integration of content-based diversity into the continuous top-k processing model presented in Chapter 3. The proposed model includes heuristics for approximate diversification and a query indexing structure for efficient processing of diversity-aware top-k queries at the social network scale.

We present the DA-SANTA (Diversity-Aware Social and Action Network Threshold Algorithm), based on continuous top-k processing model, that provides efficient continuous processing of relevance-diversity top-k queries over information streams. The DA-SANTA adopts the *max-sum diversification* bi-criteria objective function to combine relevance and diversity.

The rest of this chapter is structured as follows: section 4.1 presents the data and processing models and the adopted diversity model. In section 4.2, we introduce the DA-SANTA algorithm. In section 4.3, we present the experimental evaluation of our algorithm. Finally, section 4.4 concludes this chapter.

4.1 Data and processing models

4.1.1 Data model

We are based on the *ISSN model* presented in Chapter 3, but we only focus on the publication of a new message.

4.1.2 Relevance scoring function

We consider the scoring function presented in Chapter 3, Equation 3.3 or Equation 3.1.

4.1.3 Diversity model

We adopt the commonly used *max-sum diversification* bi-criteria objective function [GS09] to combine relevance and diversity into a single scoring function for top-k computation. In this case, the diversity of a set of results is measured by the sum of distances between the set elements. More precisely, if we note $u.TL_k = \{m_1, ..., m_k\}$ the top-k result list for user u, the diversity of $u.TL_k$ is given by the following expression:

$$D(u.TL_k) = \sum_{m_i, m_j \in u.TL_k, i<j} dist(m_i, m_j) \tag{4.1}$$

Since we consider *content-based diversification*, the commonly used measure for the distance between two messages is the complement of their content-based similarity, $dist(m_i, m_j) = 1 - sim(m_i, m_j) \in [0, 1]$. This distance is also appropriate for building a combined measure of relevance and diversity, since both measures are based on content similarity. The combined relevance-diversity score DR is a linear combination between relevance and diversity.

$$DR(u.TL_k) = \nu \; f_R(u.TL_k) + (1 - \nu) \; f_D(u.TL_k) \tag{4.2}$$

Here, $f_R(u.TL_k) = \sum_{m \in u.TL_k} rel(m, u)$ is the sum of the relevance scores in the top-k list, where rel may be a scoring function such as (Equation 3.3) or (Equation 3.1).

$f_D(u.TL_k) = \frac{2}{k-1} D(u.TL_k)$ measures the diversity score of the top-k list. The homogeneity factor $2/(k-1)$ in f_D compensates the fact that f_R sums k values, while for f_D we have $k(k-1)/2$ values.

4.1.4 Processing model

As described above, the top-k query result diversification problem is NP-hard in the general case and approximate techniques are necessary for effective query processing. In the case of streaming data, where new messages continuously arrive, the common approach is to limit the updates of the top-k query results to include only the current top-k elements, plus the new message.

In our case, for a given user u having the top-k result list $u.TL_k$, when a new message m_{new} arrives, the updated top-k list $u.TL'_k$ will be the subset of size k of $u.TL_k \cup \{m_{new}\}$ that maximizes the relevance-diversity score DR defined in (Equation 4.2):

$$u.TL'_k = argmax_{S \subset u.TL_k \cup \{m_{new}\}, |S|=k} DR(S) \tag{4.3}$$

In other words, when a new message m_{new} arrives, either $u.TL_k$ remains unchanged, or m_{new} replaces in $u.TL_k$ one of the existing messages. In the latter case, if we note m_{vic} the replaced message (the "victim"), then $u.TL'_k = u.TL_k \cup \{m_{new}\}$ - $\{m_{vic}\}$. The condition for the top-k to be updated is:

$$DR(u.TL'_k) > DR(u.TL_k) \tag{4.4}$$

Algorithm 4.1 describes this general approach, which raises two main efficiency issues. The most important one is that *the update is repeated for all the users in the social network*. Then, for each user *the top-k update method is expensive*, since it requires k computations of the DR function. Our DA-SANTA algorithm proposes solutions for both these efficiency problems.

Algorithm 4.1 Basic algorithm for updating $u.TL_k$ for all $u \in U$

On m_{new} **publication**
FORALL $u \in U$
  $m_{vic} \leftarrow argmax_{m_{vic} \in u.TL_k} DR(u.TL_k \cup \{m_{new}\} - \{m_{vic}\})$
  $u.TL'_k \leftarrow u.TL_k \cup \{m_{new}\} - \{m_{vic}\}$
  IF $DR(u.TL'_k) > DR(u.TL_k)$
    $u.TL_k \leftarrow u.TL'_k$
  ENDIF
ENDFOR

4.1.5 Problem statement

Given a social network S and a combined relevance-diversity scoring function DR such as (Equation 4.2)*, design an efficient diversity-aware user (query) pruning*

algorithm, as well as an efficient top-k update method, handling the publication of a new message in the social network.

In this perspective, we propose the processing model described by Algorithm 4.2, where the *getCandidates* function represents the user pruning algorithm, while efficient top-k update is realized by heuristically choosing a victim through the *heuristics* function. Moreover, the DA-SANTA algorithm builds on top of an existing algorithm (SANTA presented in the above chapter) for top-k continuous processing with classical relevance-only scoring. An additional challenge is to efficiently adapt an existing relevance-oriented approach to diversity-aware scoring.

Algorithm 4.2 Improved algorithm for updating $u.TL_k$ for all $u \in U$

On m_{new} **publication**
$Candidates \leftarrow getCandidates(U, m_{new})$
FORALL $u \in Candidates$
  $m_{vic} \leftarrow heuristics(u.TL_k, m_{new})$
  $u.TL'_k \leftarrow u.TL_k \cup \{m_{new}\} - \{m_{vic}\}$
  IF $DR(u.TL'_k) > DR(u.TL_k)$
    $u.TL_k \leftarrow u.TL'_k$
  ENDIF
ENDFOR

4.2 The DA-SANTA algorithm

4.2.1 DA-SANTA scoring

We consider for DA-SANTA the diversity model presented in section 4.1 and the processing approach presented in Algorithm 4.2. Then, for a new published message m_{new} and a given user u, the *heuristics* function designates $m_{vic} \in u.TL_k$ as potential victim. As shown in section 4.1, the condition for m_{new} to replace m_{vic} in $u.TL_k$ is $DR(u.TL'_k) > DR(u.TL_k)$, where $u.TL'_k = u.TL_k \cup \{m_{new}\}$ - $\{m_{vic}\}$.

We note $u.F_k = u.TL_k$ - $\{m_{vic}\} = u.TL'_k$ - $\{m_{new}\}$ the subset of k-1 results for u that do not change when m_{new} replaces m_{vic}. Then $u.TL_k = u.F_k \cup \{m_{vic}\}$ and $u.TL'_k = u.F_k \cup \{m_{new}\}$. Given (Equation 4.2) and (Equation 4.1), we obtain:

$DR(u.TL_k) = \nu\ (\sum_{m \in u.F_k} rel(m,u) + rel(m_{vic},u)) + (1-\nu)\ \frac{2}{k-1}(\sum_{m_i,m_j \in u.F_k, i<j} dist(m_i,m_j) + \sum_{m \in u.F_k} dist(m_{vic},m))$

A similar expression is obtained for $DR(u.TL'_k)$, where m_{vic} is replaced by m_{new}. By dropping the common elements, condition $DR(u.TL'_k) > DR(u.TL_k)$ becomes:

$\nu\ rel(m_{new},u) + (1-\nu)\ \frac{2}{k-1} \sum_{m \in u.F_k} dist(m_{new},m) > \nu\ rel(m_{vic},u) + (1-\nu) \frac{2}{k-1} \sum_{m \in u.F_k} dist(m_{vic},m)$

We note $D_m(X) = \sum_{x \in X} dist(m, x)$, the sum of distances from message m to messages in the set X, which could be interpreted as the *diversity of X relative to m*.

We note $dr_u(m, X) = \nu\ rel(m, u) + (1 - \nu)\ \frac{2}{k-1} D_m(X)$, which may be seen as a *simplified relevance-diversity scoring function*, combining the relevance of m for u with the diversity of X relative to m.

Then the update condition $DR(u.TL'_k) > DR(u.TL_k)$ becomes

$$dr_u(m_{new}, u.F_k) > dr_u(m_{vic}, u.F_k) \tag{4.5}$$

Note that evaluating condition (Equation 4.5) is significantly faster than for the equivalent condition on DR.

4.2.2 Victim selection heuristics

We explore two heuristics for choosing the victim message in $u.TL_k$:

1. *Minimum relevance* (MR), which selects the message with the smallest relevance to u: $m_{vic} = argmin_{m \in u.TL_k} rel(m, u)$. In the context of a relevance-only scoring function, this would correspond to the k-th element in the sorted top-k list.
2. *Minimum relevance-diversity* (MRD), which introduces a part of diversity into the heuristics. It selects the message with the smallest simplified relevance-diversity dr_u: $m_{vic} = argmin_{m \in u.TL_k} dr_u(m, u.TL_k - \{m\})$.

Remark: For both heuristics, the choice of m_{vic} is independent of m_{new}, so the victim for the next m_{new} can be selected in advance, since its value is only based on $u.TL_k$. Consequently $rel(m_{vic}, u)$, $D_{m_{vic}}(u.F_k)$ and $dr_u(m_{vic}, u.F_k)$ can be computed in advance and may change after each update of $u.TL_k$.

4.2.3 The DA-SANTA index

Like in the case of SANTA, we consider a monotonic objective function F_{DA} for the threshold-based strategy. F_{DA} is issued from the top-k update condition (Equation 4.5) and should be expressed in relation with the index dimensions.

Here $F_{DA}(m_{new}, u) = dr_u(m_{new}, u.F_k) - dr_u(m_{vic}, u.F_k)$, so given (Equation 4.5), m_{new} will enter $u.TL_k$ (by replacing m_{vic}) iff $F_{DA}(m_{new}, u) > 0$.

Given the definition of dr_u, we have:

$F_{DA}(m_{new}, u) = \nu\ rel(m_{new}, u) + (1 - \nu)\ \frac{2}{k-1} D_{m_{new}}(u.F_k) - \nu\ rel(m_{vic}, u) - (1 - \nu)\ \frac{2}{k-1} D_{m_{vic}}(u.F_k)$

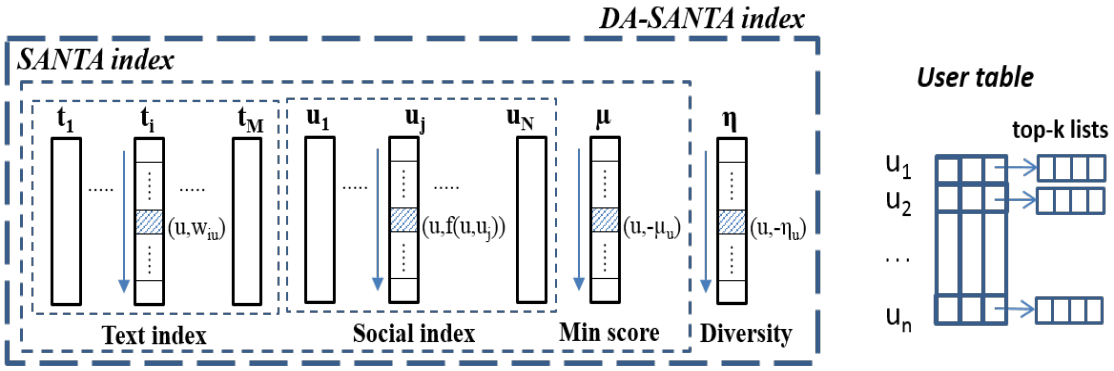


Figure 4.1: SANTA and DA-SANTA index and data structures

We note μ_u=$rel(m_{vic}, u)$ the relevance of the victim for u and η_u=$D_{m_{vic}}(u.F_k)$ the diversity of $u.F_k$ relative to m_{vic}. As explained above about the victim selection heuristics, μ_u and η_u are independent from m_{new} and can be computed in advance and maintained after each top-k update. Then the condition for m_{new} to enter $u.TL_k$ becomes:

$$\nu\ rel(m_{new}, u) + (1-\nu)\ \frac{2}{k-1} D_{m_{new}}(u.F_k) - \nu\ \mu_u - (1-\nu)\ \frac{2}{k-1}\eta_u > 0 \quad (4.6)$$

Compared with SANTA, we find in (Equation 4.6) the term in $rel(m_{new}, u)$ that corresponds to scoring functions such as (Equation 3.1) or (Equation 3.3) - this is indexed by the text and social indexes. We also have the term in μ_u, indexed by the min-score index, with the difference that the indexed value is here -$rel(m_{vic}, u)$. For the term in η_u, we add a new list η to the index (diversity index), organized like μ but storing the values of -η_u in descending order.

Therefore, the DA-SANTA index (Figure 4.1) simply adds the η list to the SANTA index structure, while the μ list contains -$rel(m_{vic}, u)$ instead of -$min_{m \in u.TL_k}(rel(m, u))$ (the k-th score).

However, the term in $D_{m_{new}}(u.F_k)$ in (Equation 4.6) cannot be indexed in a similar way. Therefore, we consider an upperbound of $F_{DA}(m_{new}, u)$, by replacing $D_{m_{new}}(u.F_k)$ with k-1, given that $D_{m_{new}}(u.F_k)$ sums k-1 distances $\in [0, 1]$. We note this upperbound F^+_{DA}:

$F^+_{DA}(m_{new}, u) = \nu\ rel(m_{new}, u) + 2(1-\nu) - \nu\ \mu_u - (1-\nu)\ \frac{2}{k-1}\eta_u$

Since F^+_{DA} is monotonic in the index entries, it can be used as an objective function for the threshold strategy. For instance, with relevance function (Equation 3.1) using cosine similarity, $score(m, u) = a\ \sum_{t_i \in m} w_{im} w_{iu} + b\ f(u, u^m) + c\ G(m)$, we obtain the following objective function, where index dimensions are underlined:

$F^+_{DA}(m_{new}, u) = \nu\ (a\ \sum_{t_i \in m_{new}} w_{im_{new}} \underline{w_{iu}} + b\ \underline{f(u, u^{m_{new}})} + c\ G(m_{new})) + 2(1-\nu) - \nu\ \underline{\mu_u} - (1-\nu)\ \frac{2}{k-1}\underline{\eta_u}$

DA-SANTA also manages a *user table* to keep information about each user in the social network. The entry for user u in the table contains:

- $u.TL_k$, the current top-k list for u;
- the necessary elements to compute $rel(m_{new}, u)$: (i) u's profile, as a list of (term, weight) couples; (ii) the list of users u' of interest for u, with the value $f(u, u') > 0$ for each one; (iii) the global importance of u in the social network, necessary to compute $G(m_{new})$ in formula (**??**);
- m_{vic}, necessary to compute $u.F_k$ and $D_{m_{new}}(u.F_k)$;
- dr_{vic}, the value of $dr_u(m_{vic}, u.F_k)$

The first component contains the current query results, while the other ones are necessary to evaluate condition (Equation 4.5) for a given m_{new}. Excepting for the elements necessary to compute $rel(m_{new}, u)$, all the other elements are updated when $u.TL_k$ changes.

4.2.4 The case of time-dependent scoring

When relevance is computed with (Equation 3.3) and uses a time bonus, the relevance-diversity score may become unbalanced because relevance values belong to [0,∞) while distances used by the diversity belong to [0,1].

To avoid this phenomenon, we normalize the relevance score to the [0,1] interval, as follows. At moment t, we divide by $TB(t-t_o)$ the relevance of a message m published at moment $t^m \leq t$, i.e. $rel_t(m, u) = tscore(m, u)/TB(t-t_o) = score(m, u)\ TB(t^m - t_o)/TB(t - t_o) \in$ [0,1].

This transforms the time factor into a time decay and makes the relevance variable in time, but the particular form of rel_t allows handling it easily. Since for a new message m_{new} its initial relevance is $score(m_{new}, u)$ (the same as for the time-independent case), nothing changes for that in the algorithm. The only impact is for the -$rel(m_{vic}, u)$ values stored into the μ list, since they correspond to older messages. The solution is to store into μ_u the relevance score with time bonus -$tscore(m_{vic}, u)$, but to multiply by an extra factor of $1/TB(t^{m_{new}} - t_o)$ the coefficient of the μ dimension in the objective function when handling the new message m_{new}. This is equivalent with considering the normalized score -$rel_{t^{m_{new}}}(m_{vic}, u)$ for μ_u.

4.2.5 The algorithm

Algorithm 4.3 presents DA-SANTA, following the framework defined by Algorithm 4.2. On publication of a new message m_{new}, the *getCandidates* method returns only users that have a chance to integrate m_{new} in their top-k. Each returned candidate is a couple (user, upperbound) - we take advantage here of the capability of the index

Algorithm 4.3 DA-SANTA algorithm

REQUIRE message m_{new}, index I, user table U
On m_{new} **publication**
FORALL $c \in getCandidates(I, m_{new})$
  $ue \leftarrow getUserEntry(U, c.user)$
  IF $c.upperbound > ue.dr_{vic}$
    $s \leftarrow$ *compute-dr*(ue, m_{new})
    IF $s > ue.dr_{vic}$
      $ue.TL_k \leftarrow ue.TL_k \cup \{m_{new}\} - \{ue.m_{vic}\}$
      $ue.m_{vic} \leftarrow heuristics(ue.TL_k)$ //MR or MRD
      $ue.dr_{vic} \leftarrow$ *compute-dr*(ue, m_{vic})
      Update $I.\mu$, $I.\eta$
    ENDIF
  ENDIF
ENDFOR

traversal method that can also estimate an upperbound for $dr_u(m_{new}, u.F_k)$ (here u is $c.user$), as described in *getCandidates*.

For each candidate, its entry ue in the user table is necessary to compute the real value of $dr_u(m_{new}, u.F_k)$. To avoid as much as possible this costly operation, we filter out cases when the upperbound is not greater than $dr_u(m_{vic}, u.F_k)$ (stored in $ue.dr_{vic}$). After computing the real score with the *compute-dr* function, if the update condition (Equation 4.5) is fulfilled, we update the top-k list $u.TL_k$, select the new victim by using heuristics MR or MRD, and update $dr_u(m_{vic}, u.F_k)$.

Finally, we update the index lists μ and η, by moving only entries for u, following the new value of -$rel(m_{vic}, u)$, respectively -$D_{m_{vic}}(u.F_k)$.

Algorithm 4.4 *getCandidates* method

REQUIRE message m_{new}, index I
initTraversal(I,m_{new}); $result \leftarrow \emptyset$; $threshold \leftarrow \overline{F^+_{DA}}(m_{new})$
WHILE $threshold > 0$
  $u \leftarrow nextIndexUser(I)$
  $result \leftarrow result \cup \{(u, \overline{dr}(m_{new}))\}$
  $threshold \leftarrow \overline{F^+_{DA}}(m_{new})$
WHILE
RETURN *result*

Algorithm 4.4 describes the *getCandidates* method that traverses the index to prune candidates. Given m_{new}, *initTraversal* selects the related lists from the index and computes the coefficients of the objective function $F^+_{DA}(m_{new}, u)$. The index lists traversal may follow any threshold algorithm strategy (e.g. TA[Fag02]) through the call to *nextIndexUser*, which returns the next user (in some of the lists) not yet seen

in the index (new candidate).

The threshold is the maximal value that the objective function F^+_{DA} may have, and is evaluated by $\overline{F^+_{DA}}(m_{new})$ as being $F^+_{DA}(m_{new}, u)$ applied to the last visited value in each index list. The monotony of F^+_{DA} and of the index lists implies that for a new candidate u, $F^+_{DA}(m_{new}, u) \leq \overline{F^+_{DA}}(m_{new})$. For the same reasons, we obtain an upperbound for $dr_u(m_{new}, u.F_k)$ through $\overline{dr}(m_{new})$, computed like $\overline{F^+_{DA}}(m_{new})$ but only on the part that corresponds to $dr_u(m_{new}, u.F_k)$. Each new candidate and its upperbound for dr_u are appended to the results list.

Also, the threshold decreases during index lists traversal, the traversal stops when $threshold \leq 0$.

Remark: The approximation of the objective function F_{DA} with its upperbound F^+_{DA} means that the index traversal finishes later, so it prunes less candidates. However, as shown in section 3.3, social network messages naturally have a good content diversity, since they contain few terms in a large dictionary. This means the difference between F_{DA} and F^+_{DA} is not important and has a reduced impact on the efficiency of DA-SANTA, as shown in section 4.3.

Theorem 2. *The DA-SANTA algorithm does not miss candidates that may be impacted by a new message.*

Proof. An user u not returned by *getCandidates* was not seen in the index before the threshold become ≤ 0. Since the threshold is the maximum value $F^+_{DA}(m_{new}, u)$ may have for an unseen u, we have $F^+_{DA}(m_{new}, u) \leq threshold \leq 0$. But $F_{DA}(m_{new}, u) \leq F^+_{DA}(m_{new}, u)$, so $F_{DA}(m_{new}, u) \leq 0$. Since the necessary and sufficient condition for u to be impacted by m_{new} is $F_{DA}(m_{new}, u) > 0$, u is not impacted by m_{new}. □

4.3 Experimental evaluation

4.3.1 Experimental setting

The social network. We extracted a subgraph from Twitter, with about $|U| = 104000$ users and $|R| = 18$ million direct links between them. The community was built starting from about 200 accounts of known French politicians and journalists, by adding part of their followers, more precisely those having a number of followees within the community above a given threshold. This method resulted into a coherent social network, with a good density of links. For the f function, we accounted also user actions such as retweet, reply, mark as favorite. The value of f depends on the existence of a direct link (u_1, u_2) and on the number of actions of u_1 on the messages of u_2. Besides the 18 million direct links, we obtained almost 1 million extra non zero values for f.

Messages, terms and user profiles. For each user we extracted the last 200 tweets. The terms considered in tweets are the hashtags, but also common and proper nouns, extracted with the TreeTagger[1] tool. We only kept non-empty messages (with at least one term) during a given period, which results in about 500 000 messages. Each message contains only a few terms, between 1 and about 10 in our corpus, with an average between 3 and 4 terms. A dictionary of about 187 000 terms was built with message terms employed by at least 5 users. For each user, the profile contains all the dictionary terms that occur in his messages. The profile size goes from 1 to around 1000 terms, with an average size of 125 terms. The weights of the profile terms, based on the *tf* and *idf* values, are computed by considering that all the messages of each user form a single document.

Relevance scoring. The relevance scoring function (Equation 3.3)(Equation 3.1) uses the default coefficients considered in the Chapter 3 (a=0.5, b=0.375 and c=0.125), which proved a good balance between the various components. For $G(m)$ we use the Klout[2] score of the message emitter, which expresses the influence of users in the main social networks, normalized to the $[0, 1]$ interval. For the time bonus we use a linear function $TB(t^m - t_o) = 1 + (t^m - t_o)/T_b$, where T_b is the period of time after which an extra bonus equal to the initial $score(m, u)$ is earned, with a default value of 15 days.

Algorithms and measures. We consider the DA-SANTA algorithm with the two victim selection heuristics, MR and MRD. For each case, we consider four combinations of factors in the relevance scoring function: *Text-Social-Time* corresponds to the complete function (Equation 3.3), *Text-Social* does not consider the time bonus and corresponds to (Equation 3.1), *Text-Time* ignores the social components considering b=c=0, and *Text* only keeps the text relevance. These combinations allow studying the influence of the various scoring components.

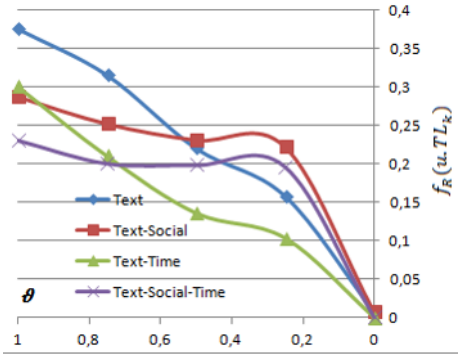
The other default values in the experiments are k=10 and ν=0.75.

We compare DA-SANTA with two other algorithms. *Baseline* corresponds to Algorithm 4.1, which evaluates each new message with all the users (queries) and tests all the possible victims in the top-k list. *Incremental* [MSN11] uses the same approach as *Baseline*, but optimizes the computation of the relevance-diversity scores by using condition (Equation 4.5) with dr_u instead of (Equation 4.4) with DR.

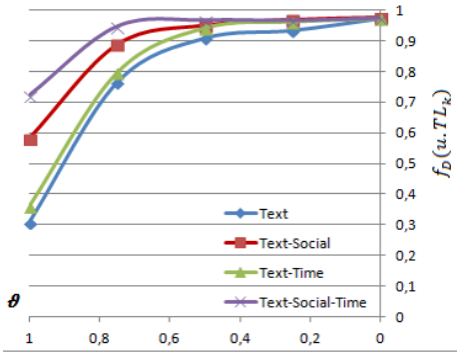
Since we aim at measuring efficiency and effectiveness in a stable status, we consider for all the algorithms an *initialization phase* that processes the first 300 000 messages, followed by *the measure phase* on the remaining 200 000 messages. All the measures (for time, relevance or diversity) represent an average over all the users and over different moments during the measure phase.

Algorithms are programmed in Java and run on a multi-core computer, in parallel with other jobs. Memory requirements for DA-SANTA are of about 1 GB for the index and 0.5 GB for the user table.

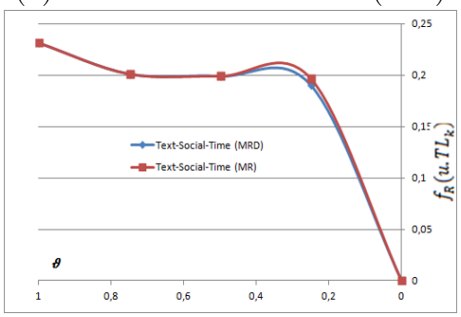
[1]http://www.cis.uni-muenchen.de/ schmid/tools/TreeTagger
[2]https://klout.com



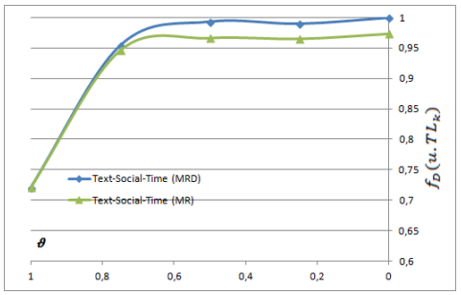
(a) Variation of relevance (MR)



(b) Variation of diversity (MR)



(c) Relevance, MRD vs MR



(d) Diversity, MRD vs MR

Figure 4.2: Variation with ν of the achieved relevance and diversity, with MR and MRD

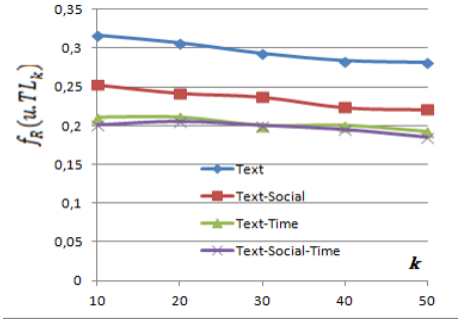
4.3.2 Effectiveness of DA-SANTA

We measure the quality of the results in terms of relevance ($f_R(u.TL_k)$) and diversity ($f_D(u.TL_k)$), depending of several parameters. The values of f_R and f_D are normalized to $[0, 1]$ by division by k.

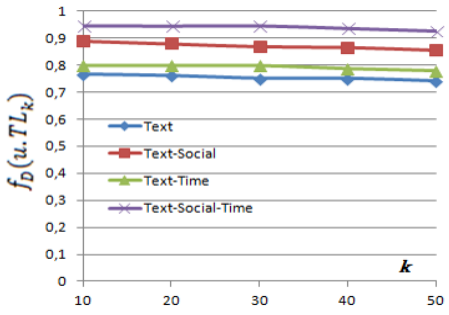
Variation with ν.

The compromise between relevance and diversity has an impact on both of them. We measure the achieved relevance and diversity, while varying the balance parameter ν. Figure 4.2 represents this variation for each type of relevance scoring, and for both victim selection heuristics, MR and MRD. Values for ν=1 correspond to the case without diversity, while ν=0 is the other extreme case, where only diversity counts.

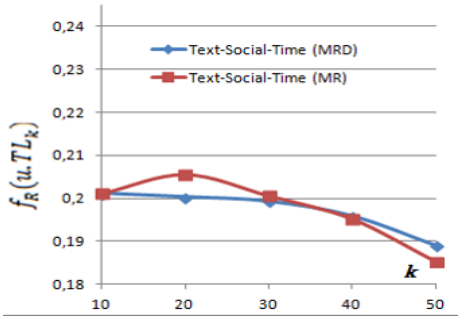
Variation of relevance (Figure 4.2.a) with MR shows a monotonic decrease of the achieved relevance in all the cases when ν decreases, to very low values when ν=0. However, when social criteria are included into the relevance scoring, the decrease is clearly much smoother. This can be explained by a better natural content-diversity of messages when the relevance is not only based on content. Note also that relevance scores are not comparable among the various scoring types, since the values are computed with different components; for instance we cannot say that time-based relevance is worse than in the corresponding cases without time, as it could appear



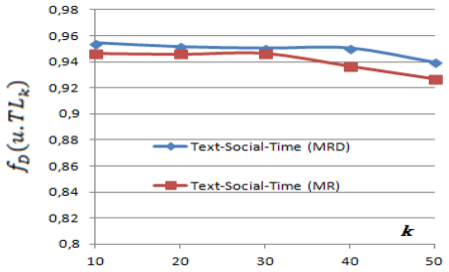
(a) Variation of relevance (MR)



(b) Variation of diversity (MR)



(c) Relevance, MRD vs MR



(d) Diversity, MRD vs MR

Figure 4.3: Variation with k of relevance and diversity, with MR and MRD

from the figure.

Variation of diversity (Figure 4.2.b) with MR shows that diversity grows when ν decreases, with a stabilization to high values around ν=0.6. Note that here, since we use the same measure for diversity, the comparison between the different cases is possible. For instance, we can see that relevance functions including more criteria provide increased content diversity. Also, the social network criteria appear to have a good influence on diversity, better than the time bonus.

When using the MRD victim selection heuristics, measures are very close to those for MR. We show only the comparison for the *Text-Social-Time* case, for relevance (Figure 4.2.c) and diversity (Figure 4.2.d), the other scoring cases providing similar conclusions. The most noticeable difference is for diversity, where, as expected, MRD provides a better score, since it considers both relevance and diversity in selecting the victim. However, the difference with MR is small.

In conclusion, a small contribution of diversity to the balance with relevance brings a very good diversity to the results, without loosing much of the relevance. Values of ν between 0.75 down to 0.5 are a very good compromise. The social relevance criteria have the best impact on diversity. We also confirm the remark in Section section 3.2 about the naturally high content diversity of social network messages, which ensures to DA-SANTA a good efficiency despite the approximation of the objective function F_{DA} with its upperbound F^+_{DA}.

Variation with k.

We measure relevance and diversity, while varying k, the size of the result set, between 10 to 50, values beyond that interval being unusual for top-k queries in social networks. Figure 4.3 illustrates the impact of this variation on relevance and diversity, for each type of relevance scoring, for MR and MRD.

Variation of relevance (Figure 4.3.a) with MR shows a slow monotonic decrease of the achieved relevance in all the cases. This is natural, since the average relevance decreases when considering more results. Diversity also decreases very slowly with k (Figure 4.3.b); this is explained by the increased probability to have similar messages in bigger result sets. The comparison with the MRD case shows a very similar behavior with the MR case for both relevance (Figure 4.3.c) and diversity (Figure 4.3.d).

In conclusion, relevance and diversity decrease with the size of the result, but the impact of k remains very limited in all the cases.

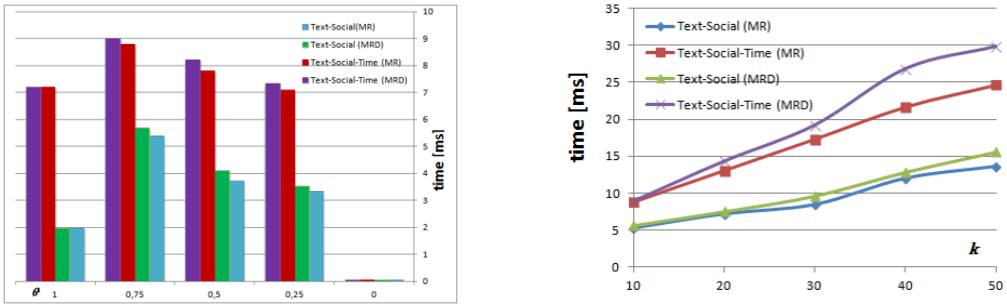
4.3.3 Efficiency of DA-SANTA

We measure the execution time per message, for both victim selection heuristics, MR and MRD. Since time-dependent scoring has a significant impact on the execution time (because of the increased probability of new messages to be relevant and to enter the top-k), we compare two scoring cases, without (*Text-Social*), and with (*Text-Social-Time*) time bonus.

Variation with ν. We measure the execution time for five values of ν: 1, 0.75, 0.5, 0.25 and 0. Figure 4.4.a presents this variation for each combination of victim selection heuristics and scoring case. In all the cases, the execution time first increases when ν decreases from 1 to around 0.75, then it decreases when ν continues to decrease. The initial increase is explained by the increasing role of the diversity in the global score, provoking more and more updates to the top-k and to the index. Around ν=0.75 the diversity becomes high enough and cannot increase too much anymore; the value of the diversity part in the objective function F^+_{DA} becomes important enough to produce a quicker termination of the index traversal. This produces less candidates, so a shorter execution time. In the extreme case of ν=0, diversity is so high that it cannot be improved most of the time, so the number of candidates is very low and the execution time close to 0.

In all the cases, MR is slightly faster than the MRD heuristics, but the difference is small. Time-dependent scoring has a much higher impact on the execution time, which is between 1.5 to 2.5 times longer with *Text-Social-Time* than with *Text-Social*.

In conclusion, the execution time of DA-SANTA is low enough (milliseconds per message) to be adapted to continuous top-k processing, even if the best quality of results (ν between 0.75 and 0.5) corresponds to the highest execution time. In similar condition, the relevance-only-based SANTA algorithm is one order of magnitude



(a) Variation with ν (b) Variation with k

Figure 4.4: Execution time for DA-SANTA, with MR and MRD

faster, but compared with diversity-aware algorithms DA-SANTA is very efficient (see below). Time-dependent scoring has a significant impact on the execution time, but do not change the order of magnitude. The victim selection heuristics has less importance here, compared to the other efficiency factors.

Variation with k. We measure the execution time for k between 10 to 50, for the same combinations of victim selection heuristics and scoring type. Figure 4.4.b indicates in all the cases a linear increase of the execution time, which remains within acceptable limits.

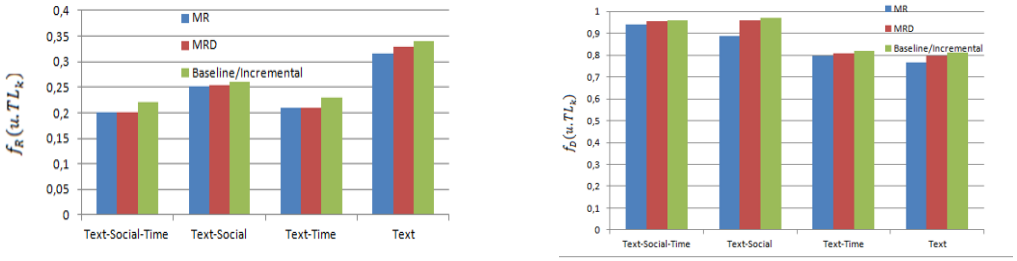
4.3.4 Comparison with *Baseline* and *Incremental*

We compare relevance, diversity and the execution time for DA-SANTA, *Baseline* and *Incremental*. Both *Baseline* and *Incremental* produce the same relevance and diversity, by testing each time all the possible victims in the top-k. Their only difference is in method of testing the relevance-diversity update condition, more efficient for *Incremental*.

Relevance and diversity. By comparing DA-SANTA with *Baseline*/*Incremental*, we evaluate the loss of relevance and diversity by applying a simpler victim selection heuristics. Figure 4.5 illustrates this comparison. We consider both MR and MRD heuristics and the various scoring cases. In all the cases, the differences are very small, which proves the very good quality of results produced by DA-SANTA.

Execution time. We compare the execution time of *Baseline* and *Incremental* with all the scoring type cases, as shown in Figure 4.6. In all the cases, *Incremental* is about 3 times faster than *Baseline*, but unlike DA-SANTA, its execution time (about 1 second/message) is not appropriate for continuous processing of top-k queries at a social network scale.

As a general conclusion, compared with *Baseline* and *Incremental*, DA-SANTA delivers a similar quality of results 2-3 orders of magnitude faster, with execution times compatible with the continuous processing of top-k queries in large social networks.



(a) Relevance (b) Diversity

Figure 4.5: Comparison with *Baseline*/*Incremental* for relevance and diversity

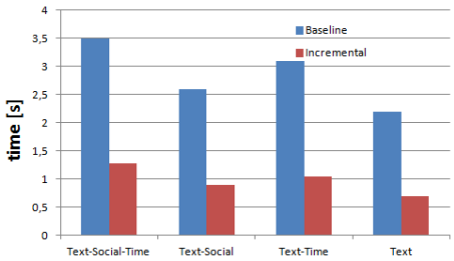


Figure 4.6: Execution time for *Baseline* and *Incremental*

4.4 Summary

In this chapter, we have focused on the query results diversity in its continuous version where the proposed solution continuously maintains diversified top-k result lists over streams of messages. Adding results diversification to continuous processing of top-k queries over information streams at a social network scale is a very challenging task. First, the general diversification problem is computationally heavy (NP-hard) and specific approximate methods are necessary, especially in the case of continuously arriving data, while preserving the quality of the results. Next, the constraint of continuous processing at the social network scale requires very efficient algorithms. Finally, when trying to extend an existing relevance-only approach to add diversity, one must face the model mismatch between top-k computation for relevance, at the element level, and diversity, evaluated at the set level. In this context, this Chapter comes with the following main contributions:

- A smooth integration of content-based diversity into the continuous top-k processing model presented in Chapter 3, which deals with information streams in a social network and uses a rich relevance function that includes content-, time- and social network-based scoring criteria. The proposed model includes heuristics for approximate diversification and a query indexing structure for efficient processing of diversity-aware top-k queries at the social network scale.
- An algorithm, DA-SANTA (Diversity-Aware Social and Action Network Threshold Algorithm), based on this model, providing efficient continuous processing of relevance-diversity top-k queries over information streams. Excepting [CC15] and complementary to it, to the best of our knowledge, this is the first

diversity-aware algorithm for continuous top-k processing adapted to large-scale social networks.

- A rich set of experiments over a real dataset extracted from Twitter, demonstrating the effectiveness and the efficiency of DA-SANTA.

5 Conclusion and Future Work

5.1 Thesis Summary

In this thesis, we deal with information streams processing, in particular, streams coming from the social networks. Messages in such information streams arrive at in a continuous manner, with high arrival rate. Users are interested in receiving new content in real time, they can express their interest by keyword queries or through an implicit query (user profile). However, the huge amount of messages certainly overloads the users with a mix of interesting and non-interesting messages. Filtering and ranking this content to allow users to be updated with only the most interesting messages is the main objective of our work. The challenges in processing information streams depend not only on the number of messages and queries, but also on the form of the ranking function. We aim at continuously processing a large number of queries defined on stream messages using a continuous top$-k$ model. In this model, the system continuously computes the k most relevant messages for each query.

The first challenge in this context is the adoption of a ranking function that includes social network factors, to improve the relevance of the results in this context. While most of the previous works in information streams use a simple scoring function, including content based and time based factors, we have proposed a richer scoring model including new social network context factors: user based and interaction based factors. In the social network, relationships between users are a good indicator to identify relevant content. Interaction with a message is a proof of its global importance, but also of the interest for the message emitter. Considering such social factors is important to improve the results quality, but it adds more complexity in the continuous computation of top-k list.

The continuous computation of the top-k query in the context of the social network is also a challenge. In addition to a large number of top-k queries, many events in the social network could update the top-k sets. Ideally, the top-k sets should be recomputed on each event in the social network (new message, new action, new edge, user profile change, ...). In our system, we consider that new messages and new actions have a strong impact on the top-k sets, and this needs to be processed immediately. The other events with a lower impact on the top-k sets could be processed periodically. To avoid processing new events with all the queries, we indexe queries with inverted indexes and use threshold techniques (such as TA) to prune queries that have no chance to update their top-k. The challenge here is to be able to include the new social network criteria into the index. As a result, we

have proposed SANTA and SANTA+ algorithms for continuous top-k processing of information streams, using a scoring function that includes the social network factors, and able to handle both events.

Our index structure is simple, flexible, easy to extend to new dimensions, unlike that of existing algorithms. They also face the problem of heavy index updates when the top-k scores change since they generally include the value of the $k-th$ score (having frequent changes) into each index dimension. In addition, they consider short queries, while social network environments come with implicit queries based on user profiles (long queries). Since the number of updates grows with the size of the query and with the number of dimensions (that increases when introducing social network criteria), these techniques are not adapted to the social network context. Unlike them, SANTA separates the k-th score from the other dimensions in the index, thus minimizing impact of updates and facilitating the extension with new social network dimensions.

Next, while existing algorithms for continuous top-k processing did not consider the results diversification problem, we addressed this problem in our work. Our goal is to provide both diverse and relevant messages in top-k sets. In this thesis, we focused on the content-based diversification where we want to maximize the dis-similarity between messages in the top-k list. The challenge is how to integrate diversity into the continuous top-k model, while maintaining the efficiency of the system. The computation of the dis-similarity between all messages in the top-k is expensive. Instead, we propose heuristics to selected a single victim message to be replaced (or not) with the new message. To this end, we have proposed the DA-SANTA algorithm that is able to continuously maintain the k messages that are both diverse and relevant for each query. The DA-SANTA algorithm follows the filtering technique employed in the SANTA algorithm by evaluating the new message with a limited number of queries.

This is the first work to our knowledge that introduces the continuous top-k model in a highly dynamic environment, the social network, with a rich scoring function and includes results diversification techniques. Our experimentation shows that the proposed algorithms (SANTA, SANTA+ and DA-SANTA) are effective for a social network with hundred of thousands of users and millions of messages. Our indexes allow a simple and efficient parallelization of the proposed algorithms to scale out to a greater social network.

However, some limitations are worth noting. Even though we have only processed the new message and a new action in continuous mode, other events deserve to be included in the continuous processing, such as changes to user profiles. Moreover, in this work, we considered only one dimension of diversity, depending on the content. Future work should explore other dimensions such as time or information sources.

5.2 Future Works

A first line of continuing this work concerns the scoring function and the way various social network factors are modeled within. For instance, in this thesis, we have considered the feedback (user actions) only through the number of actions, in order to assess the global importance of a message. However, user actions may be of several types (liking, commenting, tagging, etc.) and each type characterizes differently the published messages (commenting on a message is probably a more important indicator than liking it). On the other hand, this query-independent component is measured by considering the feedback in the global social network, but it may be also interesting to consider the feedback in the local social network of the user ("friends"). Therefore, this query-independent component may become a query-dependent component in the scoring function.

As mentioned above, in this thesis we have considered a linear scoring function with the multiplicative time bonus, but any monotonic function is compatible with our algorithms. It would be interesting to test our algorithms with other forms of monotonic scoring function. In addition, methods for the validation or learning of the coefficients of the ranking function should also be included in our system.

Another line for the future work concerns the processing model. Future work should focus on the continuous processing of other possible events in the social network. We have considered that user profiles and relationships between users are static, but in practice, they are not. User profiles are implicit queries that reflect the centers of interest. In fact, user profiles are created based on user activities that change over time. In our model, changes in user profiles results in an update in the text index presented in Chapter 3. The current text index can accept modifications in the user profiles as follows: adding a new term to the user profile leads to adding the user to a new list in the index, deleting a term leads to removing the user from the corresponding list and changing the term weights leads to changes to the order in the lists. Similarly, the relationships between users in the social network are dynamic, users create/delete connections daily. Such changes in the social graph result in an update in the social index presented in Chapter 3.

Our processing model is not affected by these changes since it adopts the time consistency setting as mentioned in subsection 3.1.4. However, the current version of our indexes needs to be optimized to support such updates in an effective way.

As explained in above, top-k results are continuously maintained for all queries (user profiles), but users generally are not connected *"online"* at all time. We are wasting time in processing every individual event for *"offline"* users. We can benefit from that point and save time by processing a group of events instead of an individual event.

Another important direction is to explore other dimensions for the diversification of results. In this work, we have only considered the textual dimension, but time, information sources or location are promising criteria to be explored. It is interesting

to propose messages in the top-k sets that come from different sources of information, occur in a different geographical area or are published in different time ranges. In our work, the definition of diversity is based on a distance function that measures the dissimilarity between the messages.

To be applied to a very large scale social network, our model should be adapted to work in a parallel environment. As mentioned in Chapter 2, our data structure allows a simple and efficient parallelization of the proposed algorithms. Our parallel strategy is encouraging and should be validated in practice.

Bibliography

[AGHI09] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying search results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, WSDM '09, pages 5–14, New York, NY, USA, 2009. ACM, http://doi.acm.org/10.1145/1498759.1498766.

[AK11] Albert Angel and Nick Koudas. Efficient diversity-aware search. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 781–792, New York, NY, USA, 2011. ACM, http://doi.acm.org/10.1145/1989323.1989405.

[BCH+03] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM '03*, pages 426–434, 2003, http://doi.acm.org/10.1145/956863.956944.

[BGL+12] Michael Busch, Krishna Gade, Brian Larson, Patrick Lok, Samuel Luckenbill, and Jimmy Lin. Earlybird: Real-time search at twitter. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ICDE '12, pages 1360–1369, Washington, DC, USA, 2012. IEEE Computer Society, http://dx.doi.org/10.1109/ICDE.2012.149.

[BKL06] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 97–104, New York, NY, USA, 2006. ACM, http://doi.acm.org/10.1145/1143844.1143857.

[BM96] Timothy A. H. Bell and Alistair Moffat. The design of a high performance information filtering system. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '96, pages 12–20, New York, NY, USA, 1996. ACM, http://doi.acm.org/10.1145/243199.243203.

[BOPY07] Christian Böhm, Beng Chin Ooi, Claudia Plant, and Ying Yan. Efficiently processing continuous k-nn queries on data streams. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 156–165, 2007, http://dx.doi.org/10.1109/ICDE.2007.367861.

[CACH14] Shiwen Cheng, Anastasios Arvanitis, Marek Chrobak, and Vagelis

Hristidis. Multi-query diversification in microblogging posts. In *EDBT*, pages 133–144, 2014.

[Cal96] Jamie Callan. Document filtering with inference networks. In *SIGIR '96*, pages 262–269, 1996, http://doi.acm.org/10.1145/243199.243273.

[CC15] Lisi Chen and Gao Cong. Diversity-aware top-k publish/subscribe for text stream. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 347–362, New York, NY, USA, 2015. ACM, http://doi.acm.org/10.1145/2723372.2749451.

[CCZ+12] Kailong Chen, Tianqi Chen, Guoqing Zheng, Ou Jin, Enpeng Yao, and Yong Yu. Collaborative personalized tweet recommendation. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 661–670, New York, NY, USA, 2012. ACM, http://doi.acm.org/10.1145/2348283.2348372.

[CG98] Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 335–336, New York, NY, USA, 1998. ACM, http://doi.acm.org/10.1145/290941.291025.

[CKC+08] Charles L.A. Clarke, Maheedhar Kolla, Gordon V. Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 659–666, New York, NY, USA, 2008. ACM, http://doi.acm.org/10.1145/1390334.1390446.

[CLOW11] Chun Chen, Feng Li, Beng Chin Ooi, and Sai Wu. Ti: An efficient indexing mechanism for real-time search on tweets. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 649–660, New York, NY, USA, 2011. ACM, http://doi.acm.org/10.1145/1989323.1989391.

[CNC11] Jilin Chen, Rowan Nairn, and Ed Chi. Speak little and well: Recommending conversations in online social streams. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 217–226, New York, NY, USA, 2011. ACM, http://doi.acm.org/10.1145/1978942.1978974.

[CNN+10] Jilin Chen, Rowan Nairn, Les Nelson, Michael Bernstein, and Ed Chi. Short and tweet: Experiments on recommending content from information streams. In *Proceedings of the SIGCHI Conference on Human*

Factors in Computing Systems, CHI '10, pages 1185–1194, New York, NY, USA, 2010. ACM, http://doi.acm.org/10.1145/1753326.1753503.

[CNPK05] Paul Alexandru Chirita, Wolfgang Nejdl, Raluca Paiu, and Christian Kohlschütter. Using odp metadata to personalize search. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 178–185, New York, NY, USA, 2005. ACM, http://doi.acm.org/10.1145/1076034.1076067.

[CSSX09] Graham Cormode, Vladislav Shkapenyuk, Divesh Srivastava, and Bojian Xu. Forward decay: A practical time decay model for streaming systems. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 138–149, Washington, DC, USA, 2009. IEEE Computer Society, http://dx.doi.org/10.1109/ICDE.2009.65.

[CZG+09] David Carmel, Naama Zwerdling, Ido Guy, Shila Ofek-Koifman, Nadav Har'el, Inbal Ronen, Erel Uziel, Sivan Yogev, and Sergey Chernov. Personalized social search based on the user's social network. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 1227–1236, New York, NY, USA, 2009. ACM, http://doi.acm.org/10.1145/1645953.1646109.

[DFMGL12] Gianmarco De Francisci Morales, Aristides Gionis, and Claudio Lucchese. From chatter to headlines: Harnessing the real-time web for personalized news recommendation. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 153–162, New York, NY, USA, 2012. ACM, http://doi.acm.org/10.1145/2124295.2124315.

[DFZN10] Elena Demidova, Peter Fankhauser, Xuan Zhou, and Wolfgang 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*, SIGIR '10, pages 331–338, New York, NY, USA, 2010. ACM, http://doi.acm.org/10.1145/1835449.1835506.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1): 269–271, December 1959, http://dx.doi.org/10.1007/BF01386390.

[DP09] Marina Drosou and Evaggelia Pitoura. Diversity over continuous data. *IEEE Data Eng. Bull.*, 32(4): 49–56, 2009.

[DP12] Marina Drosou and Evaggelia Pitoura. Dynamic diversification of continuous data. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 216–227, New York, NY, USA, 2012. ACM, http://doi.acm.org/10.1145/2247596.2247623.

[Fag02] Ronald Fagin. Combining fuzzy information: An overview. *SIGMOD Rec.*, 31(2): 109–118, June 2002, http://doi.acm.org/10.1145/565117.565143.

[FLN01] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, pages 102–113, New York, NY, USA, 2001. ACM, http://doi.acm.org/10.1145/375551.375567.

[GCK+10] Liang Gou, Hung-Hsuan Chen, Jung-Hyun Kim, Xiaolong Zhang, and C. Lee Giles. Sndocrank: document ranking based on social networks. In Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti, editors, *WWW*, pages 1103–1104. ACM, 2010, http://dblp.uni-trier.de/db/conf/www/www2010.html#GouCKZG10.

[GÖ03] L. Golab and M. T. Özsu. Issues in Data Stream Management. *SIGMOD Record*, 32(2): 5–14, 2003.

[GS09] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 381–390, New York, NY, USA, 2009. ACM, http://doi.acm.org/10.1145/1526709.1526761.

[HKC+12] Zeinab Hmedeh, Harris Kourdounakis, Vassilis Christophides, Cedric du Mouza, Michel Scholl, and Nicolas Travers. Subscription indexes for web syndication systems. In *EDBT '12*, pages 312–323, 2012, http://doi.acm.org/10.1145/2247596.2247634.

[HMA10] Parisa Haghani, Sebastian Michel, and Karl Aberer. The gist of everything new: Personalized top-k processing over web 2.0 streams. In *CIKM '10*, pages 489–498, 2010, http://doi.acm.org/10.1145/1871437.1871502.

[HMA12] Parisa Haghani, Sebastian Michel, and Karl Aberer. Efficient monitoring of personalized hot news over web 2.0 streams. *Comput. Sci.*, 27(1): 81–92, February 2012, http://dx.doi.org/10.1007/s00450-011-0178-9.

[HRT97] Refael Hassin, Shlomi Rubinstein, and Arie Tamir. Approximation algorithms for maximum dispersion. *Oper. Res. Lett.*, 21(3): 133–137, October 1997, http://dx.doi.org/10.1016/S0167-6377(97)00034-5.

[JGP+05] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 154–161, New York, NY, USA, 2005. ACM, http://doi.acm.org/10.1145/1076034.1076063.

[JW03] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 271–279, New York, NY, USA, 2003. ACM, http://doi.acm.org/10.1145/775152.775191.

[JWR00] K. Sparck Jones, S. Walker, and S. E. Robertson. A probabilistic model of information retrieval: Development and comparative experiments. *Inf. Process. Manage.*, 36(6): 779–808, November 2000, http://dx.doi.org/10.1016/S0306-4573(00)00015-7.

[KOSP11] Pavan Kapanipathi, Fabrizio Orlandi, Amit P Sheth, and Alexandre Passant. Personalized filtering of the twitter stream. 2011.

[KS12] Ali Khodaei and Cyrus Shahabi. Social-textual search and ranking. In *Intl Workshop on Crowdsourcing Web Search, Lyon, France, April 17, 2012*, pages 3–8, 2012, http://ceur-ws.org/Vol-842/crowdsearch-khodaei.pdf.

[KSJ09] Ioannis Konstas, Vassilios Stathopoulos, and Joemon M. Jose. On social networks and collaborative recommendation. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 195–202, New York, NY, USA, 2009. ACM, http://doi.acm.org/10.1145/1571941.1571977.

[LSC09] Ziyang Liu, Peng Sun, and Yi Chen. Structured search result differentiation. *Proc. VLDB Endow.*, 2(1): 313–324, August 2009, http://dx.doi.org/10.14778/1687627.1687663.

[LTG09] Kun Liu, Evimaria Terzi, and Tyrone Grandison. Highlighting diverse concepts in documents. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 545–556. SIAM, 2009.

[MBP06] Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. Continuous monitoring of top-k queries over sliding windows. In *SIGMOD '06*, pages 635–646, 2006, http://doi.acm.org/10.1145/1142473.1142544.

[MC13] Silviu Maniu and Bogdan Cautis. Network-aware search in social tagging applications: Instance optimality versus efficiency. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 939–948, New York, NY, USA, 2013. ACM, http://doi.acm.org/10.1145/2505515.2505760.

[MGD06] Alan Mislove, Krishna P Gummadi, and Peter Druschel. Exploiting social networks for internet search. In *5th Workshop on Hot Topics in Networks (HotNets06). Citeseer*, page 79, 2006.

[MP11] Kyriakos Mouratidis and HweeHwa Pang. Efficient evaluation of continuous text search queries. *IEEE Trans. on Knowl. and Data Eng.*, 23(10): 1469–1482, October 2011, http://dx.doi.org/10.1109/TKDE.2011.125.

[MSN11] Enrico Minack, Wolf Siberski, and Wolfgang Nejdl. Incremental diversification for very large sets: A streaming-based approach. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 585–594, New York, NY, USA, 2011. ACM, http://doi.acm.org/10.1145/2009916.2009996.

[PGC+10] Tim Paek, Michael Gamon, Scott Counts, David Maxwell Chickering, and Aman Dhesi. Predicting the importance of newsfeed posts and social network friends. In *AAAI*, volume 10, pages 1419–1424, 2010.

[PvA08] Krešimir Pripužić, Ivana Podnar Žarko, and Karl Aberer. Top-k/w publish/subscribe: Finding k most relevant publications in sliding time window w. In *DEBS '08*, pages 127–138, 2008, http://doi.acm.org/10.1145/1385989.1386006.

[RBS10] Davood Rafiei, Krishna Bharat, and Anand Shukla. Diversifying web search results. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 781–790, New York, NY, USA, 2010. ACM, http://doi.acm.org/10.1145/1772690.1772770.

[RCCT14] Weixiong Rao, Lei Chen, Shudong Chen, and Sasu Tarkoma. Evaluating continuous top-k queries over document streams. *World Wide Web*, 17(1): 59–83, January 2014, http://dx.doi.org/10.1007/s11280-012-0191-3.

[SB88] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5): 513–523, August 1988, http://dx.doi.org/10.1016/0306-4573(88)90021-0.

[SCK+08] Ralf Schenkel, Tom Crecelius, Mouna Kacimi, Sebastian Michel, Thomas Neumann, Josiane X. Parreira, and Gerhard Weikum. Efficient top-k querying over social-tagging networks. In *SIGIR '08*, pages 523–530, 2008, http://doi.acm.org/10.1145/1390334.1390424.

[SCZ09] Aaron R Sun, Jiesi Cheng, and Daniel Dajun Zeng. A novel recommendation framework for micro-blogging based on information diffusion. 2009.

[SGFJ13] Alexander Shraer, Maxim Gurevich, Marcus Fontoura, and Vanja Josifovski. Top-k publish-subscribe for social annotation of news. *Proc. VLDB Endow.*, 6(6): 385–396, April 2013, http://dx.doi.org/10.14778/2536336.2536340.

[TSZ06] Bin Tan, Xuehua Shen, and ChengXiang Zhai. Mining long-term search history to improve search accuracy. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 718–723, New York, NY, USA, 2006. ACM, http://doi.acm.org/10.1145/1150402.1150493.

[VAC12] Nelly Vouzoukidou, Bernd Amann, and Vassilis Christophides. Processing continuous text queries featuring non-homogeneous scoring functions. In *CIKM '12*, pages 1065–1074, 2012, http://doi.acm.org/10.1145/2396761.2398404.

[vLGOvZ09] Reinier H. van Leuken, Lluis Garcia, Ximena Olivares, and Roelof van Zwol. Visual diversification of image search results. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 341–350, New York, NY, USA, 2009. ACM, http://doi.acm.org/10.1145/1526709.1526756.

[Vou15] Nelly Vouzoukidou. *Continuous top-k queries over real-time web streams*. PhD thesis, University Pierre et Marie Curie, September 2015.

[VRB+11] Marcos R Vieira, Humberto L Razente, Maria CN Barioni, Marios Hadjieleftheriou, Divesh Srivastava, Caetano Traina, and Vassilis J Tsotras. On query result diversification. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1163–1174. IEEE, 2011.

[XXWL13] Xiaokui Xiao, Yabo Xu, Lingkun Wu, and Wenqing Lin. Lsii: An indexing structure for exact real-time search on microblogs. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, ICDE '13, pages 482–493, Washington, DC, USA, 2013. IEEE Computer Society, http://dx.doi.org/10.1109/ICDE.2013.6544849.

[YBLS08] Sihem Amer Yahia, Michael Benedikt, Laks V. S. Lakshmanan, and Julia Stoyanovich. Efficient network aware search in collaborative tagging sites. *Proc. VLDB Endow.*, 1(1): 710–721, August 2008, http://dx.doi.org/10.14778/1453856.1453934.

[YGM94a] Tak W. Yan and Hector Garcia-Molina. Index structures for information filtering under the vector space model. In *ICDE'94*, pages 337–347, 1994, http://dl.acm.org/citation.cfm?id=645479.655112.

[YGM94b] Tak W Yan and Héctor García-Molina. Index structures for selective dissemination of information under the boolean model. *ACM Trans. on Database Syst. (TODS)*, 19(2): 332–364, 1994.

[YGM99] Tak W. Yan and Hector Garcia-Molina. The sift information dissemination system. *ACM Trans. Database Syst.*, 24(4): 529–565, December 1999, http://doi.acm.org/10.1145/331983.331992.

[YLAY09] Cong Yu, Laks Lakshmanan, and Sihem Amer-Yahia. It takes variety to make a world: Diversification in recommender systems. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*,

EDBT '09, pages 368–378, New York, NY, USA, 2009. ACM, http://doi.acm.org/10.1145/1516360.1516404.

[YLL10] Peifeng Yin, Wang-Chien Lee, and Ken C. K. Lee. On top-k social web search. In *CIKM*, pages 1313–1316. ACM, 2010, http://dblp.uni-trier.de/db/conf/cikm/cikm2010.html#YinLL10.

[ZC01] Yi Zhang and Jamie Callan. Maximum likelihood estimation for filtering thresholds. In *SIGIR '01*, pages 294–302, 2001, http://doi.acm.org/10.1145/383952.384012.

[ZMKL05] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 22–32, New York, NY, USA, 2005. ACM, http://doi.acm.org/10.1145/1060745.1060754.