Continuous top-k queries in social networks

Abdulhafiz Alkhouli, Dan Vodislav, and Boris Borzic

ETIS - ENSEA / Univ. of Cergy-Pontoise / CNRS - France
{abdulhafiz.alkhouli,dan.vodislav,boris.borzic}@ensea.fr

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. Top-k queries over the streams of interest allow limiting results to the most relevant content, while continuous processing of such queries is the most effective approach in large scale systems. However, current systems fail in combining continuous top-k processing with rich scoring models including social network criteria. We present here the SANTA algorithm, able to handle scoring functions including content similarity, 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. We compare SANTA/SANTA+ with an extension of a state-of-the-art algorithm and report a rich experimental study of our approach.

Keywords: information streams, social networks, continuous top-k query processing, publish/subscribe systems

1 Introduction

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. Yet, 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 filtering and ranking techniques which allow users to efficiently be updated with the most interesting items.

Ranking models for information streams proposed so far mainly focus on the item content and/or publication time, by laying on scoring functions based on two categories of factors: (1) *content-based*, measuring the adequacy of the item content with the subscription query, and (2) *time-based*, measuring the decrease of the importance of an item as time goes by.

However, *the cooperative dimension* of such environments has been neglected in the ranking models. The importance of an item depends not only on its content or publication time, but also on the users, on the relationships between them and on their actions. The relationship between publishers of information streams

and subscribers introduces *a social network dimension* in this publish-subscribe (pub-sub) framework. This dimension includes not only *explicit* relationships between users in a social network, but also *implicit* ones, through the interaction of users with items published by other users (e.g. likes, shares, etc.). This social network dimension provides new criteria to measure the interest of items for users, in addition to those mentioned above: (3) *user-based*, measuring e.g. the importance of the publisher and of the relationship between the subscriber and the publisher in the social network graph, and (4) *interaction-based*, measuring the importance of items by the reaction they provoked, expressed through actions of other users on that item. As many studies show, including these criteria into the scoring function improves the quality of the ranking [21][10].

We address here the problem of filtering the large amount of items through *top-k continuous queries*, where the ranking model is based on a scoring function that includes social network criteria. An important challenge in this context is the design and implementation of *efficient processing models* at a very large scale. The main difficulty comes from the need of continuously computing (or re-computing) the score of every item 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 items and queries, but also on the form of the scoring function.

We distinguish two main categories of processing models proposed to date. *The static approach* is based on periodic snapshot queries over the set of published items to get the top-k list 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 lists of important messages. 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 [8][14], most of the time based on the textual content, possibly combined with time factors, while more complex scoring models, including social network factors are only handled through the static approach.

Our purpose is to go beyond the state-of-the-art methods for *continuous processing of top-k queries* over information streams, by considering *complex scoring functions that include various factors related to the social network environment.* Also, unlike existing continuous approaches [8][11][16] that only handle new item publication events, we aim at *extending continuous processing to interaction with items* (action events). In this paper we propose the following main contributions:

- 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.
- A variant of SANTA, called SANTA+, that significantly accelerates the processing of action events.

– 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.

2 Related Work

Ranking models for information streams. In ranking models for documents, the importance of a document for a subscription query has been generally considered in the context of text documents and queries, based on information retrieval (IR) text relevance models such as tf-idf [15] or Okapi BM25 [9]. To this query-dependent score model, some approaches have also added a *global, query independent importance of messages*, based on the PageRank score [12] when messages refer web pages, on information novelty [7], on source authority [5] or on user attention [19].

The social network context has been considered in the scoring models, in order to improve the relevance of subscription query results by taking into account the relationships between publishers and subscribers. Social network components are included in the score model in several approaches, such as the distance in the social graph [1] [21], user actions [10] or location information [20].

However, the complexity of these scoring models prevented their use for continuous top-k processing. Either they are only proposed to provide a better relevance estimation in social network environments, or, at best, they come with efficient algorithms for some score components computation (e.g. distance in graph) and with static, snapshot-based algorithms for top-k evaluation [20] [21]. 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 [18], but this is limited to the simplest component, a global, query independent importance of each message. Very recently, [17] reported an extension of this work including user feedback in the score model, which can be assimilated to our interaction-based score components.

In this context, our work aims at proposing a rich score model, including social network components, providing a good compromise between expressiveness and a complexity suitable for continuous top-k processing.

Continuous top-k processing of information streams. The large majority of the methods proposed for continuous top-k processing consider text information streams, with text subscription queries, IR relevance models (tf-idf, BM25) and no social network scoring elements.

Part of them consider the time-based aspects by using a time-sliding window w (top-k/w pub-sub), where items exiting the window become irrelevant. Among them, [13] is an early work on probabilistic models for continuous top-k processing, while [11] considers classical tf-idf cosine similarity, using two inverted text indexes, one for the most recent messages (in the sliding window) and the second one for the subscription queries. Top-k processing is based on the Threshold Algorithm (TA) [6] exploiting the text indexes. However, since messages are indexed, a high arrival rate results here in expensive index updates.

A well known work in this category is [8], which proposes the COL-Filter algorithm and an improved variant POL-Filter. COL-Filter only indexes subscription queries but uses a score-oriented order for the inverted lists instead of query-oriented order in [11]. More precisely, an index list for a query term τ contains queries q that include τ, ordered by the ratio between the importance of τ in q and the current k-th best score for q. This allows efficient top-k processing by using the Threshold Algorithm (TA) [6] on the index lists, but suffers from a relatively high number of updates subsequent to k-th best score changes provoked by new messages or by message exit from the time sliding window.

In a similar context, [14] proposes a strategy for sharing effort among queries in the top-k computation process, based on a covering relationship between subscription queries and an associated graph index. However, their covering relationship is not adapted to the extension with social network criteria.

Other approaches replace the time sliding window with a continuous order-preserving decay function to handle time-dependent scoring, which eliminates the problem of top-k re-computation upon message expiration. In this category, [16] proposes an adaptation of two IR top-k retrieval strategies to information streams: the document-at-a-time (DAAT) algorithm WAND [2] and the term-at-a-time (TAAT) algorithm of Buckley and Lewit [3]. However, the approach looks difficult to extend with social network scoring criteria.

Unlike the above approaches considering text information streams with monotonic and homogeneous scoring functions, [18] introduces a global importance of each message in scoring, that may be computed on social network criteria. They use a two-dimensional inverted query indexing scheme and use decay functions for time-dependent scoring, like [16]. As mentioned above, this work has been extended [17] with the inclusion of user feedback in the score model and in event processing, but user relationships in the social network are not considered.

All these continuous top-k techniques are not extensible to include social network criteria, excepting COL-Filter [8] and [18]. Even for them, 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.

The rest of the paper is organized as follows: the next section describes the data model, the scoring function and the processing model, then Section 4 presents the SANTA and SANTA+ algorithms. Finally, Section 5 presents the experimental study, before concluding.

3 Data and processing models

Data model. We consider information streams produced by the users of a *social network*. Streams are composed of *messages* (items), each message being char-

acterized 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.*
- *$p : U \to \mathcal{D}$ is a function associating a profile to each user. User profiles and message contents are both modeled as content descriptors in $\mathcal{D}$.*
- *$sim : \mathcal{D}^2 \to [0, 1]$ measures the similarity between two content descriptors.*
- *$f : U^2 \to [0, 1]$ 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.*
- *$s : U \to \mathcal{I}$ 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 graph, 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, such as the paths connecting u_1 to u_2 in the graph, the actions of u_1 on the messages of u_2, 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.

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.

Scoring function. We consider here a scoring function that expresses $score(m, u)$, 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.

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

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. We consider a *decay function* [18][16], $TD : \mathbb{R}_+ \to [0, 1]$, 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 $t \geq t^m$:

$$tscore(m, u, t) = score(m, u) \cdot TD(t - t^m) \tag{2}$$

Here $score(m, u)$ is the scoring function from (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 [4]. 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:

$$tscore(m, u, t) = score(m, u) \cdot TB(t^m - t_o) \tag{3}$$

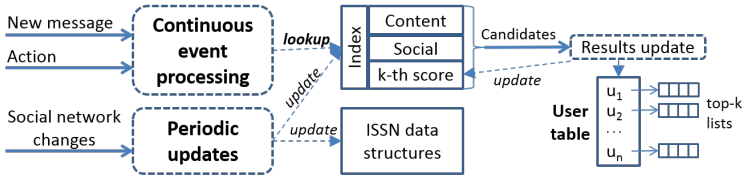


Fig. 1. Model of continuous top-k processing in an ISSN

Problem statement. *Given an ISSN and a scoring function such as* (1) *or* (3)*, design an algorithm that efficiently computes and maintains the lists of best k messages for each user, as new messages are published and new actions on the existing messages are registered.*

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. For space reasons, no exhaustive study of handling the local impact of social network changes is presented, but only a brief discussion in Section 4.

Figure 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.

4 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

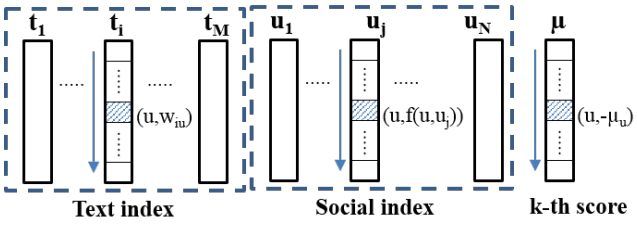


Fig. 2. The SANTA index structure

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 [8].

Index and other data structures. The SANTA index structure (Figure 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 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.

Scoring function. We consider the case of a scoring function such as (1) or (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 (1) can be written:

$$score(m, u) = a \sum_{t_i \in m} w_{im} w_{iu} + b\, f(u, u^m) + c\, G(m) \tag{4}$$

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 (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 [6] to traverse the index lists in order to get candidates u for top-k change.

In the case of the time-dependent scoring function (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.

The algorithm. Figure 3 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 *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 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 [6], 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 $threshold \leq 0$.

```
Algorithm SANTA
Input: message m, action a,
       index I, user table U
  newMessage (m, I, U)
    getCandAndUpdate(m, I, U)
  end
  newAction (a, I, U)
    m ← getMessage(a)
    getCandAndUpdate(m, I, U)
  end
  getCandAndUpdate (m, I, U)
    foreach c in getCandidates(I,m) do
      ue ← getUserEntry(U,c.user)
      if c.upperBound > ue.kthScore then
        //compute real score
        s ← computeScore(m,ue)
        if s > ue.kthScore then
          update ue and I.μ
      end if
    end foreach
  end
  getCandidates (I, m)
    initTraversal(I,m)
    result ← ∅; threshold ← F̄(m)
    while threshold > 0 do
      u ← nextIndexUser(I)
      result ← result ∪ (u, scorē(m))
      threshold ← F̄(m)
    end while
    return result
  end
```

```
Algorithm SANTA+
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
```

Fig. 3. The SANTA and SANTA+ algorithms

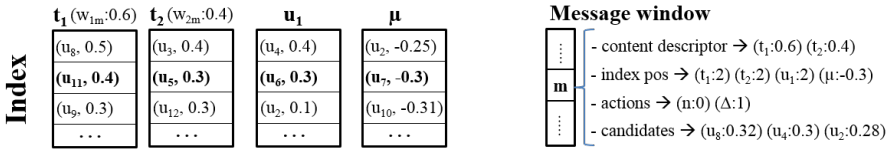


Fig. 4. Execution example for SANTA and SANTA+

Figure 4 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 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·$\Delta G(m)$=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·(0.6·*0.3*+0.4·*0.3*) + b·*0.1*+c·0.2-*0.31* = -0.09 < 0.

SANTA+: improving action processing. 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 Section 3, $AI(m) \in [0, 1]$ 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.

Figure 3 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*. In the case where the message has exited the window, it uses the SANTA algorithm through *getCandAndUpdate*.

Figure 4 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 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 (Section 3), 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 consistency. 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 (u_1,u_2) requires the insertion of u_1 into the social index of u_2 with some default importance, etc.

CF+: an extended version of COL-Filter. As mentioned above, the closest approach to ours is the COL-Filter algorithm [8], 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 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.

5 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 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.

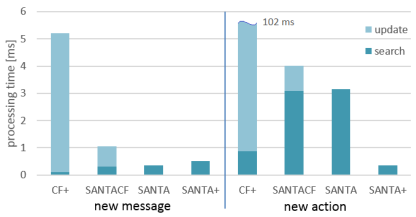
Social relations. For the values of the $f(u_1, u_2)$ function in the ISSN, we combined only two factors: the existence of a direct link (u_1, u_2) and the number of actions made by u_1 in relation with u_2. Besides the 18 million direct links, we obtained almost 1 million extra non zero values for f. 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, with no con-

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



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

Fig. 5. Comparison between CF+, SANTA and SANTA+

sequence on the conclusions. For most experiments we use the whole dataset, with messages and actions processed in timestamp order.

For space reasons we do not study here the variation of α, β and γ. Measures confirm a good choice for the default values and show a good stability of the processing time with α and γ, while increasing β values produce a significant speedup, but favor too much messages from important users in the network.

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.

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.

The left part of Figure 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 [8] 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. We measure besides the execution time also 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.

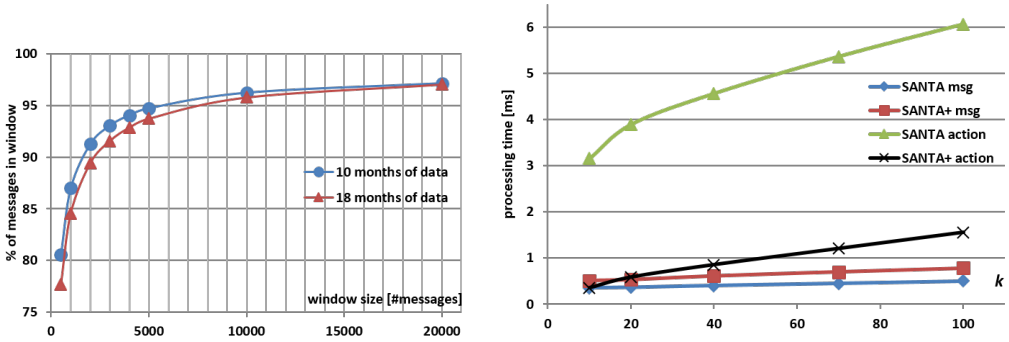
The measures, in the table on the right side of Figure 5, correspond to an average over the test dataset. *In the no time bonus case* measures are similar to those from the comparison with CF+. For message processing, SANTA+ is slower than SANTA, which is explained by the message window management, but 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.

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-



(a) (b)

Fig. 6. (a) The impact of a bounded message window; (b) Variation with k

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 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 6(a) 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.

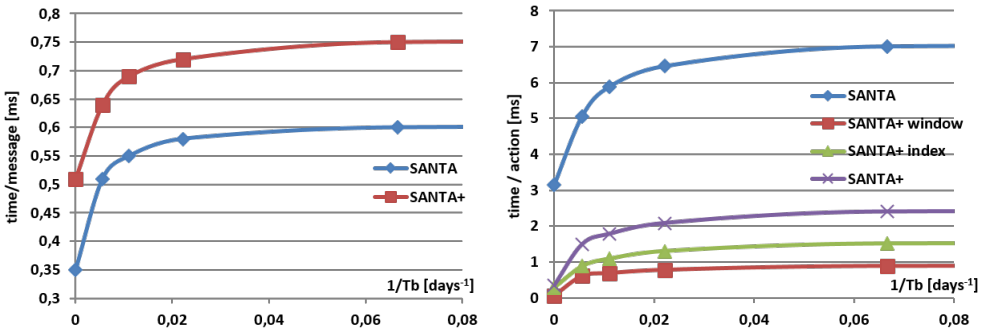


Fig. 7. Varying the time bonus for message and action processing

Varying *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 6(b) 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 7 presents a finer analysis of this phenomenon for message (on the left) and action processing (on the right), 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 processing, 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.

6 Conclusion

This paper presented SANTA, the first algorithm for continuous top-k processing over information streams in a social network context, able to handle both message publication and user actions on existing messages, with a rich scoring function mixing content-based, time-based, user-based and interaction-based components. The experiments show that SANTA and its variant SANTA+, which optimizes action processing, are an effective solution for extending continuous top-k processing of information streams with various social network dimensions. Future work will consider the extension to other events and other scoring components, as well as results diversity.

References

1. B. Bahmani and A. Goel. Partitioned multi-indexing: Bringing order to social search. In *WWW '12*, pages 399–408, 2012.
2. A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM '03*, pages 426–434, 2003.
3. C. Buckley and A. F. Lewit. Optimization of inverted vector searches. In *SIGIR '85*, pages 97–110, 1985.
4. G. Cormode, V. Shkapenyuk, D. Srivastava, and B. 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.
5. G. M. Del Corso, A. Gullí, and F. Romani. Ranking a stream of news. In *WWW '05*, pages 97–106, 2005.
6. R. Fagin. Combining fuzzy information: An overview. *SIGMOD Rec.*, 31(2):109–118, June 2002.
7. E. Gabrilovich, S. Dumais, and E. Horvitz. Newsjunkie: Providing personalized newsfeeds via analysis of information novelty. In *WWW '04*, pages 482–490, 2004.
8. P. Haghani, S. Michel, and K. Aberer. The gist of everything new: Personalized top-k processing over web 2.0 streams. In *CIKM '10*, pages 489–498, 2010.
9. K. S. Jones, S. Walker, and S. E. Robertson. A probabilistic model of information retrieval: Development and comparative experiments. *Inf. Process. Manage.*, 36(6):779–808, Nov. 2000.
10. A. Khodaei and C. Shahabi. Social-textual search and ranking. In *Intl Workshop on Crowdsourcing Web Search, Lyon, France, April 17, 2012*, pages 3–8, 2012.
11. K. Mouratidis and H. Pang. Efficient evaluation of continuous text search queries. *IEEE Trans. on Knowl. and Data Eng.*, 23(10):1469–1482, Oct. 2011.
12. L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford University, 1999.
13. K. Pripužić, I. P. Žarko, and K. Aberer. Top-k/w publish/subscribe: Finding k most relevant publications in sliding time window w. In *DEBS '08*, pages 127–138, 2008.
14. W. Rao, L. Chen, S. Chen, and S. Tarkoma. Evaluating continuous top-k queries over document streams. *World Wide Web*, 17(1):59–83, Jan. 2014.
15. G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, Aug. 1988.
16. A. Shraer, M. Gurevich, M. Fontoura, and V. Josifovski. Top-k publish-subscribe for social annotation of news. *Proc. VLDB Endow.*, 6(6):385–396, Apr. 2013.
17. N. Vouzoukidou. *Continuous top-k queries over real-time web streams*. PhD thesis, University Pierre et Marie Curie, September 2015.
18. N. Vouzoukidou, B. Amann, and V. Christophides. Processing continuous text queries featuring non-homogeneous scoring functions. In *CIKM '12*, pages 1065–1074, 2012.
19. C. Wang, M. Zhang, L. Ru, and S. Ma. Automatic online news topic ranking using media focus and user attention based on aging theory. In *CIKM '08*, pages 1033–1042, 2008.
20. D. Wu, Y. Li, B. Choi, and J. Xu. Social-aware top-k spatial keyword search. In *MDM '14*, pages 235–244, 2014.
21. P. Yin, W.-C. Lee, and K. C. K. Lee. On top-k social web search. In *CIKM*, pages 1313–1316. ACM, 2010.