Real-Time, Scalable, Content-based Twitter users recommendation

Julien Subercaze, Christophe Gravier, Frédérique Laforest

Real-Time, Scalable, Content-based Twitter users recommendation

Julien Subercaze [a], Christophe Gravier [a] and Frédérique Laforest [a]
[a] *Laboratoire LT2C, Equipe Satin, Telecom Saint-Etienne, 25 rue du docteur Remy Annino, 42000 Saint-Etienne, France*
E-mail: first.last@telecom-st-etienne.fr

Abstract.
Real-time recommendation of Twitter users based on the content of their profiles is a very challenging task. Traditional IR methods such as TF-IDF fail to handle efficiently large datasets. In this paper we present a scalable approach that allows real time recommendation of users based on their tweets. Our model builds a graph of terms, driven by the fact that users sharing similar interests will share similar terms. We show how this model can be encoded as a compact binary footprint, that allows very fast comparison and ranking, taking full advantage of modern CPU architectures. We validate our approach through an empirical evaluation against the Apache Lucene's implementation of TF-IDF. We show that our approach is in average two hundred times faster than standard optimized implementation of TF-IDF with a precision of 58 %.

Keywords: Twitter recommendation, Binary Footprint, Large scale approach, Information Retrieval, Real-Time recommendation

1. Introduction

Microblogging websites such as Twitter produce tremendous amounts of data each second. For instance, Twitter was known to publish an average of 140 millions tweets per day as of march 2011[1]. In a single year, this number has increased up to 340 millions tweets[2]. In this context, we address the problem of building a "good" user profile model, in order to exploit it in applications such as recommending users to users. We expect a "good" user profile model that maximizes the following characteristics :

- *Distance-preserving* : The distance between users in the user profile space should preserve the interesets proximity perceived by users with their peers as much as possible.
- *Explanability* : Recommender systems should be able to produce human-understandable justifications of recommendations.
- *Extensibility* : The set of users and their interests evolve with time. However, it should be possible to update the model without complete reprocessing.
- *Scalability* : Algorithms to compute and compare user profiles should present a complexity as low as possible.
- *Real-Time* : The profile must be often updated, there its computation must be as fast as possible. These algorithms should also be easily parallelized, to take advantage of the advent of Map/Reduce [9] and related paradigms.

In this paper we present a new approach to create and compare profiles of social network users. The solution exploits user-generated contents. In this model, we generate a binary footprint – *a hash* – of the user profile that preserves the distance

[1] `http://blog.twitter.com/2011/03/numbers.html`
[2] `http://blog.twitter.com/2012/03/twitter-turns-six.html`

between users profiles in the binary space. Using a binary footprint provides both scalability and parallelization for computing and comparing user profiles. Computing Hamming distance between two hashes is a very fast operation that is computed at the processor level [3] on commodity machines. Moreover, computing newcomers profiles does not require to recompute others' profiles.
The paper is organized as follows. Section 2 presents related works on building user profiles. Section 3 describes the core of our proposition, which is a complete processing of user-generated contents resulting in building and comparing user profiles. Details on how we deal with finding representative data structures in the user profile is provided in section 3.1. In Section 4, we present architecture design and implementation details of the storage and retrieval database used to perform top-k retrieval on hashed profiles. We apply this approach on a Twitter dataset that we crawled, and compete with other content-based techniques, taken from the field of Information Retrieval. Experimental results validate the quality and the real-time property of our approach in section 6. Section 7 concludes and provides hints for further investigations.

2. On building social user profiles

Twitter analysis has been applied for folksonomies homogenisation [45], tag recommendation [37,44] or as a corpus for opinion mining and sentiment analysis [30,14]. It is often used as a datasource for recommender systems. Among these systems a distinction is done between user recommendations and user-generated content recommendations. User-generated content recommendations concern tweets or external content sources suggestions to users whereas users recommender systems suggest users to follow as an output. Both rely on a digital representation of each user's features, which is usually called the user's profile. The field of recommender systems blossomed with the advent of Web-scale data analysis, especially with applications in Web search and social networks. While it has grown so as to be now considered as a new data science by itself, we tried to meekly cover how building user's profile is performed in the myriad of existing works. User profile can also be used in personalization systems [11,7]. Exhaustive literature reviews with better coverage can be found in [1], [18], and [24]. To our knowledge, user recommender systems fall in the following schools of thought: Social Network Analysis, Collaborative Filtering, Semantic-based models, and Content-based models. They can also combine these approaches for building hybrid recommender systems. Our proposal (see Section 3) belongsto the content-based model category.

2.1. Social Graphs

A possible approach to build user profiles is to consider information from the social network of the user. The assumption is that the proximity of users in a social graph conveys the distance between users' interests, hence the user recommendation scheme. The proximity between users mainly relies on concepts such as graph density and centrality in order to identify key users in the social graph. In these works, the problem is not taken as ranking all possible friends to a given user. Instead, it most frequently aims at finding clusters of users (sometimes referred as communities) in the social graph. Recommended users are then picked within the user's cluster. For instance, in [8], the authors introduced a users segmentations method using the Gaussian Mixture Model (GMM). In [19], the authors look for hubs and authorities in the network and then identify possible overlapping and dense communities in the network.

The major issues with social networks were stressed by Social Network Analysis studies [43]. They demonstrate that content and user interactivity prevail over the social graph of the user. Moreover, note that most of the algorithms applied to social graphs are NP-hard problems, and even applying heuristics or approximations are at best in polynomial time.

2.2. Collaborative Filtering

Collaborative Filtering is a very popular research area for recommender systems. It builds recommendations based on past item-user interactions, previous ratings of items by users. It is

[3] Recent processors with the SSE4 instruction set allow the computation of hamming distance in two operations, `XOR` and `POPCNT`

primarily used in item recommender systems [36], as competitions are organized around large publicly available datasets, like the Netflix dataset [4]. They can also be used for user to user and item to item recommender systems. State of the art implementations mainly rely on Matrix Factorization methods [21].

Collaborative filtering methods are known to provide good results given enough data. They suffer from the infamous *sparse data* and *cold start* problems. Computation is also at stake when the matrix is in a order of magnitude of billions of lines and columns. However, given both users and items are projected in a feature space, they can provide some justifications for the recommendation they made, for instance by illustrating the recommendation with close users-items interactions in the feature space [21]. Collaborative Filtering can be improved using demographic data [41].

2.3. Semantic-based models

Semantic-based models exploit external databases in order to enrich user profiles, especially linked data. In [28], the authors presents Flink, a system that builds user profiles out of users web pages, emails, and even FOAF profiles. FOAF is a cornerstone for these approaches, since it is the *de facto* standard for modeling user profiles in the Semantic Web. This category of user profiles usually complement Collaborative Filtering, as they address the cold start issue. Nonetheless, they does not scale well, because reasoning over logic fragments involved in the Semantic Web is, even for minimalist fragments like [29] in polynomial time, and at worst NEXPTIME [39]. Regarding the user recommendation process that follows user profile construction, [38] proposed the use of semantic technologies for better people recommendation in a system called *Social Adviser*. The authors introduced linked data (DBPedia, a semantized version of Wikipedia) in the content extraction process. They have also defined specific scores to measure expertise. Even if no performance analysis is provided, [17] showed that executing SPARQL queries over the Web of data takes at least 20 seconds even with all data locally retrieved in advance, which discards *de facto* such an approach for real-time purpose.

2.4. Content-based models

Content-based models aim at modeling users with predefined features. For instance, users can present geolocation, gender, age..., while items could have features such as metadata, topics, hashtags, etc. There are many possible features and ways to combine them, thus leading to as many different models. For instance, [25] builds user profiles out of folksonomies. From the user personomy, a bipartite graph is built and a greedy algorithm looks for clusters of tags, and most frequent tags in the cluster serve as a signature of this cluster.

A large spectrum of works tries to leverage the textual information that were produced or rated by the user. Our proposal that follows in Section 3 falls in this category. Early popular approaches rely on Vector Space Models (VSM) [35]. VSM uses frequency measures in text corpus in order to leverage semantic information. Lingras used modified Kohonen self-organizing maps to cluster web users [23], which can be lately used for recommandation purpose.

Frequencies encompass [40] :

- frequencies of terms in documents, for modeling user interests based on their choice of words.
- frequencies of terms in the context of a given term for word sense disambiguation.
- frequencies of word pairs with other word pairs, in order to model latent word pairs semantic relatedness.

Most of these techniques use bags of words to characterize users or documents [34,15]. As the bag of words approach has shown its limitations, machine learning techniques have been developed to go beyond this representation. The most popular are continuous Conditional Random Fields (CRF) [22] and Latent Dirichlet Association [5] (LDA). Both techniques have been used to represent documents [42,32] and LDA has also been used for topic modeling in social network [33]. The main drawback of these machine learning techniques is the learning part, which is prohibitively extensive for real-time processing. Wallach [42] provides mean execution time for LDA. Results show that each iteration (around 200 are required) takes between 2 and 15 seconds. Although presenting computational issue, these approaches extract knowledge automatically,

	Distance-Preserving	Extensibility	Scalability	Parallelizable	Explanations
Social Graphs	−	+	−−	−	+
Collaborative Filtering	+	−	+	++	+
Semantic-based	+	−	−−	−	+
Content-based	+	−	−−	++	−
Hybrid	++	−−	−−	+	−−

Table 1
Pro/cons of differents schools of trends for building user profiles as an input to recommender systems.

thus avoiding the hurdle to create *ad hoc* ontologies like in semantic-based approaches.

2.5. Hybrid methods

Several works tried to combine the aforementioned approaches. For instance, [20] combines Social graph with Collaborative Filtering. They propose Referral Web, a Web-scale system for searching social networks for users and items at both coarse and fine grain. In [10], the authors propose a combination of collaborative filtering and content-based methods for item recommendation. Another combination is Content-based mixed with Collaborative Filtering as proposed by [27] for improved recommendations. In [16], Twitter users are recommended some followers using content and Collaborative Filtering approaches. Hybrid systems are gaining momentum since they have proven to beat single approach baseline in large and popular dataset. In lessons learnt from the Netflix price in [3], the authors stress the importance of using a variety of models that complement the shortcomings of each others. This is however beyond the scope of this paper. We focus on a proposal falling into the Content-based category, and expect that improving one of these categories would improve the overall performance of hybrid recommender systems. We provide a synoptic view of the different schools of trends on building user profile in Table 1, in the light of the criteria presented in Section 1 that we want to maximise when building a user profile.

As a conclusion, the real time processing of textual data cannot be handled efficiently with the use of external data sources or with machine learning techniques.

In this paper we developed a document centric approach, with the use of statistics and graph techniques. Given shortcoming stressed in this section and gathered in Table 1, the aim is to keep the advantages of Content-based approaches but provide them extensibility and scalability using heuristics and hash functions. This approach is described in the next section.

3. `HashGraph` - Binary user profiles

Our approach called `HashGraph` is inspired by the paper of Matsuo & Ishizuka [26] on keyword extraction. In this article the authors present a method to extract keywords from a single document using statistical information. In a first step their algorithm computes the co-occurency matrix of the terms in each sentence of the document. Then they apply the following procedure "Co-occurrences of a term and frequent terms are counted. If a term appears selectively with a particular subset of frequent terms, the term is likely to have an important meaning. The degree of bias of the co-occurrence distribution is given by the χ^2-measure." The χ^2-measure is later used in their algorithm to extract relevant keywords or keyphrases, the other words of interest being discarded.

We adapt the approach of Matsuo & Ishizuka for user profile extraction on twitter. A document is the set of users' tweets. In our approach we aim at keeping all relevant words in the users' profiles, with corresponding weights. We adapt the approach by skipping the χ^2 value computation which is highly computationnaly expensive. And we replace this method by a clustering by a strong component detection from the graph of terms.

This computation can be done efficiently in linear-time.

The main steps for Twitter user recommendation based on content, using our `HashGraph` approach, are the following:

1. Grab text from user tweets so as to build a document representing the user
2. Preprocess the document
3. Build the graph of terms
4. Compute the hash of the profile
5. Identify the k nearest neighbors and recommend them to the end user

Figure 1 (inspired by [46] to compare keyword extraction processes) summarizes this process. The whole process must be computable in real-time, so as to react "instantaneously" to the evolution of users tweets topics.

Some of these steps do not deserve much attention. In the following we focus on two main actions. The first action is to transform a document containing a sequence of short text messages into a graph of terms. We present the method for this transformation in section 3.1. Since we focus on a real time algorithm, we discuss the complexity of the algorithms used. The second action concerns hashing user profiles and computing a distance between hashed profiles. It is the object of section 3.2.

We define an undirected weighted graph of terms $G =< V, E >$ where V is the set of vertices (i.e. the terms) and E is the set of edges. E is an application from V to V. To each $e \in E$, a weight w is associated $:w \in \Re$, where w is the proximity between the two terms.

3.1. User profile as graphs of terms

We process the tweets using standard text analysis steps. Instead of sentences split, our algorithm considers a tweet as a sentence, i.e. all the words in a tweet are co-occurring, even if they belong to two different sentences in the same tweet. The idea being to consider a tweet as a unity in terms of performative speech act. The terms are then stemmed using a standard Porter-stemming algorithm. From the set of terms, we restrict the set of candidate terms for building the graphs to the lemmatized tokens and extracted n-grams minus the stopwords. Part of speech tagging is used to extract only nouns, verbs and adjectives from the tweets. Afterwards we build the co-occurrence matrix of the terms based on a tweet split. Table 2 presents an example of a co-occurrence matrix.

Once the co-occurrence matrix is built, we transform it into a graph representation. Several approaches are possible. The naive approach would consider the co-occurrence matrix as an adjency matrix of the graph. We have decided to consider rows of the matrix as an occurence probability distribution and to compare distributions of terms using statistical divergence measure.

	a	b	c	d	e	Total
a		3	4	2	1	10
b	3		0	0	2	5
c	4	0		4	0	8
d	2	0	4		6	12
e	1	2	0	6		9

Table 2
Co-Occurrence matrix example for terms a,b,c,d and e

If we normalize the values of each row in the matrix so that the sum is equal to one and assuming that terms are independently occurring, we can consider the rows of the co-occurrence matrix to be a distribution probability. This means that for row of term a, each cell represents the probability that term a co-occurs with another one. Table 3 presents the normalized values for the first row.

	a	b	c	d	e	Total
frequency		3	4	2	1	10
probability		0.3	0.4	0.2	0.1	1

Table 3
Co-occurence frequency and probability for the term a

For instance probability of co-occurrence of a and c is 0.3. To determine wether two terms belong to a same topic, it is possible to evaluate the distance of their co-occurrence probability distribution. Several statistical measures provide either divergence or metrics, we used the square root of the Jensen-Shannon divergence.

Very different terms have a low Jensen-Shannon divergence. They are not of interest in our approach, thus we set a threshold to select the interesting value. A commonly adopted threshold is $0.95 \times log(2)$ [26]. We ran several tests on our

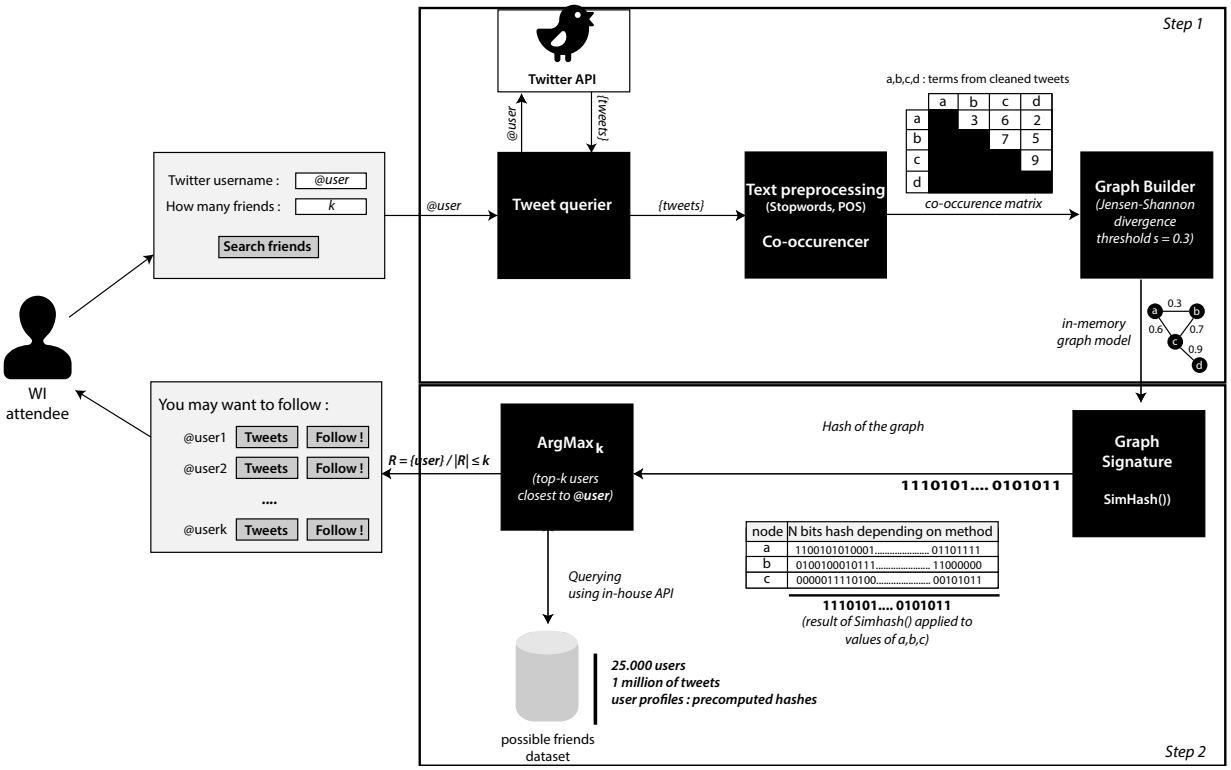


Fig. 1. Successive steps of the algorithm.

dataset and agreed with this value. We then build the graph of the terms using the selected values. The results for the co-occurrence matrix of our example are depicted in figure 2.

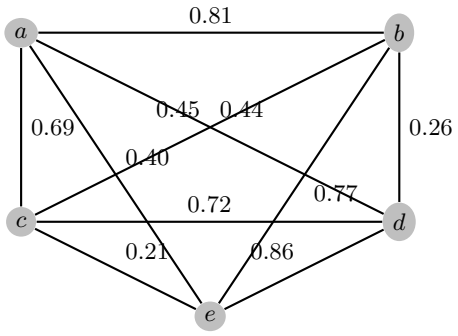


Fig. 2. Graph of terms example using table 2 co-occurrence matrix

Computing the divergence between two discrete probability distributions of N events requires $\frac{N\times(N-1)}{2}$ operations for comparison between rows times $N-1$ comparisons between each p_i and q_i. This complexity seems prohibitive but in fact co-occurrence matrices are very sparse; consequently the number of operations is drastically reduced. Analysis from the dataset are presented in Table 4. In the average only 7.5% of the values are defined in the matrix. This seems logical since not few terms co-occur with other terms. We observe that users with few tweets (≤ 25) have more dense matrix than users that have much more tweeted. This means that when the matrices are dense, their size is very small, thus the total computation time remains low. We also noticed that users that tweet very much (≥ 400) and that have abnormal small and dense matrices are users that generate automatically their content such as services providing regularly news from their website using a few templates to generate their tweets. For example a web dictionary that would tweet "'The word FOO has been defined, check out http://mydict/FOO"' each time a new word is defined, it would be affected a small and dense matrix. Consequently one could use the deviation to the average of the co-occurrence matrix in order to detect spam and automatically generated content.

Definition 1 (Jensen-Shannon Divergence) *For two discrete probabilities P and Q, the Jensen-Shannon*

divergence is given by the following formula :

$$JSD(P||Q) = log(2) + \frac{1}{2}(D(P||M) + D(P||M))$$

where $M = \frac{1}{2}(P + Q)$*. and* D *is the Kullback-Leibler divergence. The Jensen-Shannon divergence is comprised between 0 and 1 :* $0 \leq JSD(P||Q) \leq 1$*.*

Corollary 1 (Metric) *The square root of the Jensen-Shannon Divergence is a metric [13].*

Optimization To speed-up the graph computation, it is possible to take advantage of the fact that the square root of the JS divergence is a metric (Corollary 1). Considering geometrically the relations between three points a, b, c where the distance between a and b and between b and c are known, it is possible to obtain an upper bound for the distance between a and c. Since the square root of Jensen-Shannon is a metric, it verifies the triangle inequality i.e. $|a, c| \leq |a, b| + |b, c|$. Considering the fact that we discard distances inferior to a given threshold, we can use the upper bound from the triangle inequality to know without calculating if the distance will be sufficient. If the upper bound is smaller than the threshold, then calculating the distance would be superfluous. In the worst case, if all the distances are over the threshold then the gain is null. In the best case, the two third of the distances helped to avoid the computation of the last third. Thus the maximum reduction is of $N/3$ operations. In practice, we observed a gain around 8%, which is not to be neglected when dealing with large amount of data.

3.2. Hashing user profiles and distance between profiles

Using the technique presented before, each user can have his/her profile modeled as an undirected weighted graph, whose vertices are terms extracted from the user's tweets. We then require to compute a compact footprint of this graph, that can be used for both storage and comparison. We have investigated different options to encode a graph as a bit array, so that the Hamming distance could be used as a similarity metric between user profiles, and can also be highly prone to be inserted in a Map/Reduce implementation [9].

Sparsity Average	0.075
Sparsity standard derivation	0.15

Table 4

Sparsity of the co-occurrence matrix - Based on our tweets dataset

We found a representative hash function of our user profile modeled as a graph a terms. An important review and some approaches on hashing graphs can be found in [31]. [6] proposed SimHash, a hash function for generating a footprint out of a graph. SimHash can be applied to any kind of resource (document, images ...), and in our case a graph.

In SimHash, the resource, usually a document, is splitted into token, possibly weighted. Each token is then represented as its hash value, as the result of a traditional cryptographic function applied to the token, which is originally a string. Then, a vector V, of length of the desired hash size, is initialized to 0. For each hash value for the set of tokens, the i^{th} element of V is *decreased* by the corresponding token's weight if the i^{th} bit of the hash value is 0. Otherwise, the i^{th} element of V is *increased* by the corresponding token's weight. Figure 3 depicts an example of SimHash for three tokens. SimHash works well even for small fingerprints [6], and was historically applied to the detection of near-duplicates of web crawl graphs.

We use SimHash to compute binary user profiles by hashing graph of terms with the following settings:

- As SimHash features : the set of edges and vertices of the graph of terms.
- As Simhash edges weights : the normalized Jensen-Shannon divergence values, which is the edge's weight.

Nodes and edges must be manipulated as a bit array in SimHash. The hash values, depending on the hash function used are very compact. For example, with the MD5 algorithm, user profiles are 128 bits long, thus allowing to store 64 millions profiles in 1GB. Using MD5, it allows us to manage 820 billions profiles with a collision probability around 10^{-15}. We tested several hash functions, as presented in our evaluation in the next section.

Finding users to recommend is thus solved as getting the top-k closest hashed profiles using a Hamming distance, which has the property to be rapidly computed and easily parallelizable.

Token	Weight	Hash
a	3	1 0 1 1 0 1
b	2	0 1 1 0 0 1
c	1	1 0 0 1 1 1

$\Longrightarrow$

Token	Weight	Hash
a	3	3 -3 3 3 -3 3
b	2	-2 2 2 -2 -2 2
c	1	1 -1 -1 1 1 1

$\Downarrow$

Token	Weight	Hash
a	3	3 -3 3 3 -3 3
b	2	-2 2 2 -2 -2 2
c	1	1 -1 -1 1 1 1
total		2 -2 4 2 -4 6
hash		1 0 1 1 0 1

Fig. 3. SimHash example for tokens {a,b,c} with respective weights {3,2,1}.Each token is assigned its hash values, then 0 bits are replaced by -1, then multplied by the weight of the token, and a final sum is computed

4. Hash Database

Retrieving top-k related profiles efficiently is the key to obtain great performance on the overall system. In this section we detail the architecture design and implementation details of the internal database.

The database stores key/value pairs, where the key is the hash of the profile and the value if the Twitter username related to the hash. The database offers two methods:

Push(hash/username) this methods adds a new pair into the database.

TopK(hash,k) this methods retrieves the top-k profiles for a given hash. This hash is called the *query*.

To compute the top-k method, the system must scan the entire database to compute distances between the query and all the elements in the database. To perform this operation efficiently, we store the hashes into a flat array of N-bits objects – N is the number of bits of a hash. Each hash is identified by its index in the array and Twitter username are stored in a dictionary that maps username to index in the array. Using a flat array to store hashes allows to take advantage of locality of reference while scanning the data. This enables the hardware prefetcher to perform optimally, fetching in advances the next hashes to store them in the CPU cache [12]. For large number of hashes, we can split the array to parallelize the operation.

After the distance computation, the top-k values are retained, these values are the indices of the k closest hashes in terms of Hamming distance. Finally, the list of top-k usernames is built looking up the dictionary.

4.1. Storing distance

Computing the distance between two hashes is a very fast operation using modern computers intrinsics. For each comparison query/i-th hash, the result is a pair of key/value, where the key is the distance and the value is the index of the hash in the database. A naive approach is to store these key/value and to extract the top-k once the computation of distance between query and hashes is terminated.

To improve performance, we propose an alternative design. Our idea is based upon a practical observation: On one hand, the number of bits N of a single hash is small (i.e. ≤ 1024), therefore the distance between two requires $log_2(N)$ to be stored. On modern 64-bit architecture, most of the bits would be unused: $log_2(1024) = 10$. On the other hand, the database is unlikely to store 2^{64}= 18 quintillion profiles. Since there are only 7 billion people on the planet (even with multiple ac-

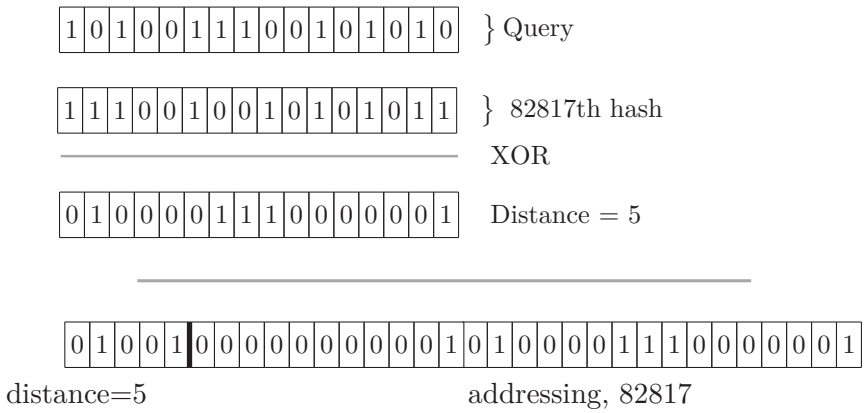


Fig. 4. Storing address and distance in a single word. The query hash and the 82817th hash are xored to compute the Hamming distance. The final result containing the distance and the address is stored into a single 32-bit words. For readability reasons, the example is given with 16-bit hashes and 32-bit words.

counts). For instance, on 1024-bit hashes, we can use 10 bits to store the distance between hashes and the rest 54 bits to store the index of documents – $2^{54} = 18$ million billions, is largely sufficient as an adress space.

Thus, we split the 64-bit of a single word into two parts : $log_2(N)$ most significant bits to store the distance and the remaining $A = 64 - log_2(N)$ as an address space.

Figure 4 shows an example of such a storage of distance/adressing into a single word.

4.2. Older vs Newer first

While comparing distance for top-k retrieval using our single word storage system, we introduced de facto a secondary order. The first ordering takes place on distance, which is stored on the most significant bits. The second ordering is on the index of the nodes. For instance, we continue the example presented in Figure 4. Let admit the 82818th hash also has a Hamming distance of 5 and there are already k values into the top-k datastructure and 5 is the highest distance among these values. When the distance with the 82818th is computed, the result of the comparison distance will be higher than the one of the previous hash, due to the index. As a consequence, this approach will favor (at equal distance) older results.

To circumvent this issue and to offer the choice to favor the newer result, we introduce a modification into the storage system. So that the last shall be the first, we change the encoding of the index. We use the two's complementary of the bitarray representing the adress. Thus, small number become large and vice versa. To compute this complementary value, we compute a `AND` operation between the adress and a mask with the A least significant digits set to 1.

4.3. Top-k datastructure

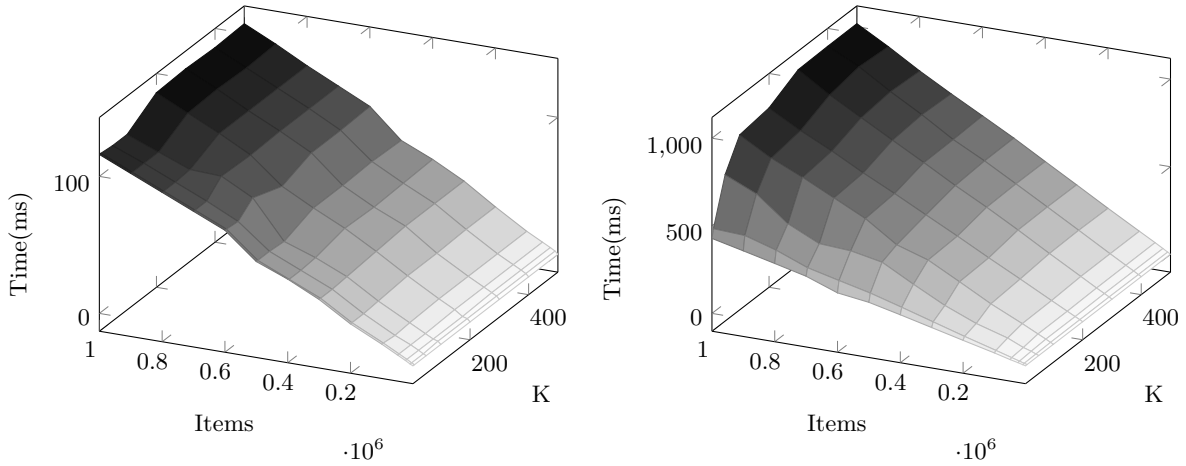
We consider two methods for the top-k retrieval. The first method is store all the distances between the query and the hashes into a list, to sort this list and retain the k first values. The second method uses a Min-Max Heap [2]. This structure is a double-ended priority queue. It allows to find the min and max value in constant time, while insert and delete min/max are performed in logarithmic time.

4.4. Field testing

To determine the performance of our system, we evaluated our hash database with both top-k datastructure. The database is implemented in Java, we used Google Guava's double-ended priority queue implementation [4].

Array list based structure was implemented by us. To harness the performance of the structure, we proceed to a battery of tests . For each test, we

[4] `http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/collect/MinMaxPriorityQueue.html`



(a) Array List (b) MinMax Heap

Fig. 5. Comparison of top-k datastructure performance on 128-bit hashes

populate the database with n items – ranging from 1000 to 1 million. We query 10 times the database for top-k values on against a random query. The paramater k varies from 5 to 500.

The results of the tests are depicted in Figure 5. The value of k has naturally no influence on the array list approach. The influence on the Min-Max Heap is more important, especially when the number of items is large. On pure running time, to our surprise, the simplest solution, the array list, with the worst theoretical complexity, consistently presents the best performance. It outperforms the Min-Max Heap by a factor 10 for 1 million items.

5. Implementation

We implemented our prototype in the Java language. To represent the binary signatures we extended the `BitSet` class to fix the length of the signature. Computing the Hamming distance for large dataset on modern processors can be achieved using very fast processor intrinsics.

To compute the Hamming distance between two binaries signatures, one simply `XOR` the two bitsets and count the bits set to one in the resulting bitset.

In the CPU instruction set Streaming SIMD Extensions 4.2 [5], the instruction `POPCNT` allows to count the number of bits set to one in a word. All the tests conducted in this paper were done on a Intel core i7-3720QM implementing SSE 4.2. We ensured that the assembly generated by the Hotspot JVM used intrinsics for Hamming distance computation.

[5] http://software.intel.com/sites/landingpage/IntrinsicsGuide/

6. Experimental results

6.1. Building dataset

Twitter users share messages called tweets that are limited to 140 characters. Using the public API of Twitter it is possible to retrieve tweets of any user that has not set its profile private (which is in practice very rare). A users's sequence of tweets is commonly called the user's timeline. To create a user's profile, one could choose either to retrieve all the tweets from the timeline or to select a subset from this timeline. A common intuition is that all the tweets, especially the very old ones may not be of great interest, then selecting a subset would be appropriate. To ensure our choice, and in order to determine the most interesting subset, we conducted some tests on our dataset. In case of analyzing a subset, we then have to determine how to limit this subset : should it be time limited or bounded by a number of tweets.

We first crawled up to 1K tweets for 5K users. We then analyzed how many tweets users actually wrote, and their distribution among time. Table 5 presents some basic analysis of the collected dataset.

Average tweets per user	≈ 120
Standard deviation of tweets per user	≈ 212
Average interval between two tweets	≈ 6 *days*
Standard deviation between two tweets	≈ 240 *days*

Table 5
Tweets distribution

Distribution over time is biased by numbers of accounts that produced a few tweets at account creation and later remain unused. However the distribution over time is not a very precise indicator. There is a very broad range of usage patterns. One may tweet regularly whereas one other may tweet once in a while. Therefore it is very difficult to setup a relevant time window.

The average tweets number per user is suprisingly low on the sample, however the standard deviation being higher than the mean is an indicator of high volatility of the measure.

Figure 6 presents the distribution of users whose number of tweets is comprised into intervals. The distribution of the users appears to be a Maxwell-Boltzmann distribution. This similarity is confirmed by looking at the cumulative distribution depicted in Figure 7. The cumulative distribution is indeed a square root distribution (Table 7).

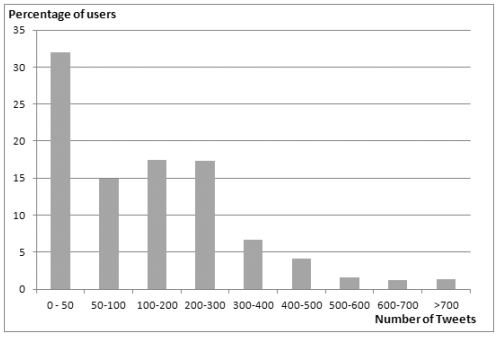


Fig. 6. Repartition of users into tweets intervals

The spatial distribution of the graphs in a scatter plot is given in Figure 8.

The volatility of the results shows that once again it is hard to decide whether we should limit the subset on a temporal or on a numerical basis. Since a decision had to be made, our choice went to a numerical basis, for the simplicity and stability of data harvest. From the cumulative results, we were able to select the number of tweets per user to extract, in order to cover the largest set of users. Technically the Twitter API provides 200 tweets per page. That's why we splitted into interval of 200 tweets – we present the distribution in table 6.;

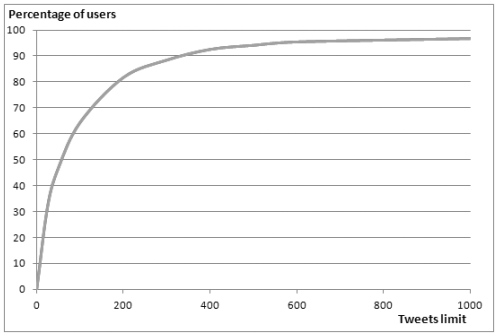


Fig. 7. Percentage of users having less than a given number of tweets

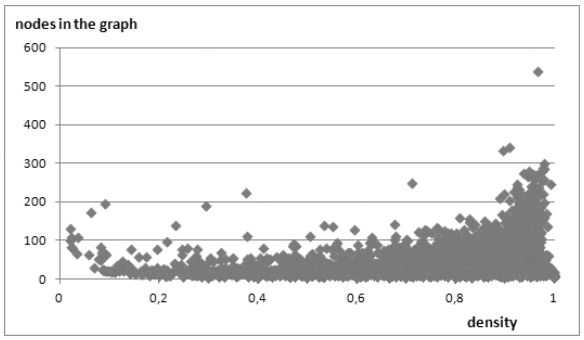


Fig. 8. Distribution of the users' terms graphs

Limits	Percentage of users
200	81,6%
400	92,48%
600	95,43%
800	96,25%

Table 6
Percentage of users having less than a given number of tweets

Requests to the Twitter API are limited and time-costly. Since the gain between 600 and 800 is less than 1%, we decided to set the limit at 600 tweets per user, which seemed to us to be an acceptable trade-off.

We then gathered about 1 million tweets for around 25K users. The tweets are stored on a three

machines Cassandra-cluster[6]. Due to the current Twitter's terms of services we are unfortunately not able to publicly provide this dataset.

6.2. Runtime and Quality

We compared our approach against standard Vector Space Model approach. For this purpose, we implemented our solution in Java without any particular optimisation. As a baseline we used the Lucene[7] based TF/IDF vectors with the cosine similarity.

Although containing only 25K Twitter users, our dataset contain more than one million terms in the Lucene index, therefore the curse of dimension would have disastrous consequence on cosine similarity computation with such high dimensional vectors. Therefore, we precomputed TF/IDF vectors for various number of frequent terms, we also precomputed hashes using our approach with three cryptographic hash methods : MD5, SHA-256 and SHA-512. Precomputation time are depicted in figure 10. The difference of precomputation time for 25K users between TF/IDF and our approach `HashGraph` (20 seconds vs 110 seconds) could be overcomed by implementation optimisation. One should not forget that we compare an optimized, in-production solution (Lucene) with a research prototype. This difference should not obscure that the average time required to precompute a binary user profile is only 5 milliseconds and could be reduced with an optimized implementation.

The running times for the four methods are shown in figure 9. The x axis shows the number of distance comparisons performed. The slight decrease at the beginning of the `HashGraph` curves is counter-intuitive and is to our opinion, to be attributed to the Java JIT compiler warm up phenomenon.

The `HashGraph` algorithm clearly outperforms the cosine similarity by three orders of magnitude. For instance, to compute 10 millions comparisons between profiles, the cosine similarity requires 150 seconds, while `HashGraphMD5` requires only 253 milliseconds. As one would expect from simple complexity analysis, the running time grows linearly in the number of comparison for both methods.

The computation time for comparisons does not grow linearly for 128, 256 and 512 bits. This is mainly due to prefetching and caching effects of the processor. Since we traverse linearly memory blocks containing the binary footprints, the memory access pattern is easy to detect for the processor, thus enhancing global performance.

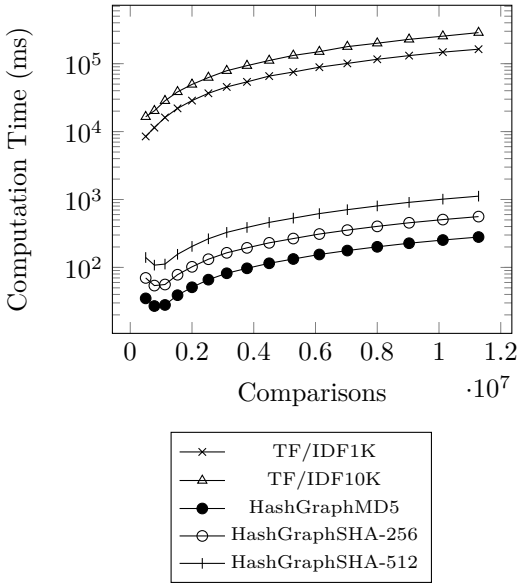


Fig. 9. User profiles distance : running time of TF/IDF and HashGraph

To determine the quality of our approach against the baseline, we use the standard root-mean-square error (RMSE) on pairwise distances between users on the dataset. The distance between two users u_i, u_j in the baseline is the standard cosine similarity denoted $cos(u_i, u_j)$. Let v_i, v_j being two TF/IDF vectors representing users u_i and u_j. The similarity is defined as follows :

$$sim(u_i, u_j) = \frac{v_i \times v_j}{||v_i||.||v_j||}$$

The distance between two hashes is the normalized version of the standard hamming distance.

We computed the precision on the set of users using the following RMSE :

$$1 - \sqrt{\sum_{i=0}^{N} \sum_{j=i}^{N} (sim(u_i, u_j) - hamming(u_i, u_j))^2}$$

[6] `http://cassandra.apache.org/`
[7] `http://lucene.apache.org/`

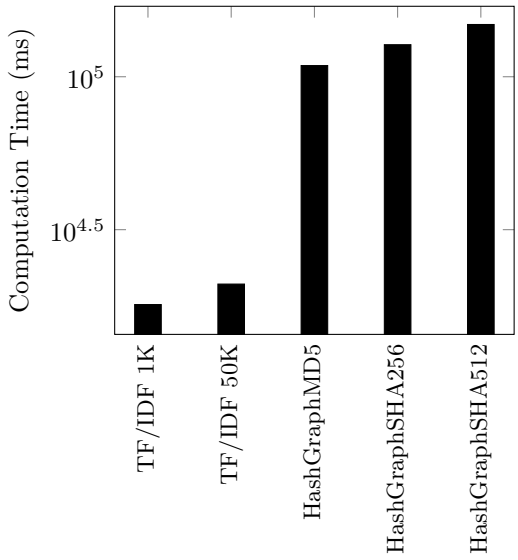


Fig. 10. Precomputation time

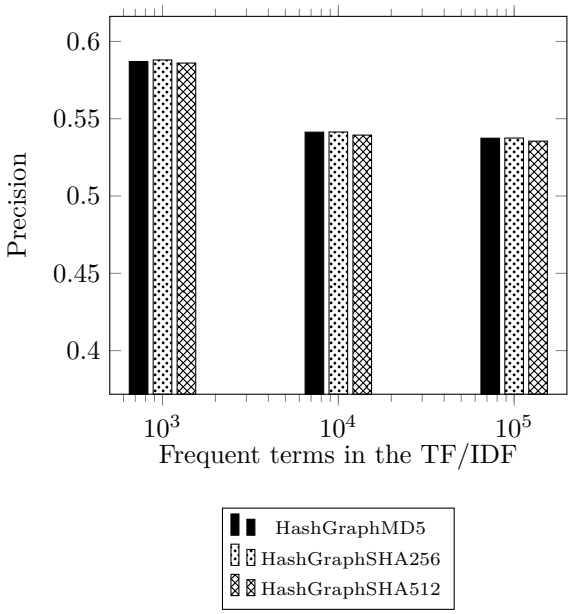


Fig. 11. Precision (1-RMSE) of the different *HashGraph* functions against TD/IDF with various frequent terms

The obtained precision, regardless of the number of frequent terms used in the cosine similarity distance is greater than 0.5. It also appears that the choice of hash function in simhash does not have much influence on the precision. Since MD5 provides the most compact representation with 128 bits, it will be our method of choice.

7. Conclusion

In this paper we presented `HashGraph`, a global system for real-time Twitter user recommandation. We built compact binary user profile using tweet contents. For this purpose, we used the graph derived from the terms cooccurrence matrix as input for SimHash. We use a dedicated optimized hash database to perform efficient top-k retrieval. The computed precision in this paper allows us to conclude that the results obtained with `HashGraph` are coherent with state of the art content based user profile. Our system is orders of magnitude faster than the baseline, and is suitable for real-time recommendation.

Further research will aim to determine the perceived quality of recommandation through user evaluation, as well as enhancing the descriptive power of our approach by focusing on the design of a dedicated family of hash functions.that should provide better semantic distance preservation than the currently used cryptographic functions.

References

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[2] M. D. Atkinson, J.-R. Sack, N. Santoro, and T. Strothotte. Min-max heaps and generalized priority queues. *Commun. ACM*, 29(10):996–1000, October 1986.

[3] Robert M. Bell and Yehuda Koren. Lessons from the netflix prize challenge. *SIGKDD Explor. Newsl.*, 9(2):75–79, December 2007.

[4] James Bennett, Stan Lanning, and Netflix Netflix. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[5] D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[6] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. *Approximation Algorithms for Combinatorial Optimization*, pages 139–152, 2000.

[7] Chih-Ming Chen. Incremental personalized web page mining utilizing self-organizing hcmac neural network. *Web Intelligence and Agent Systems*, 2(1):21–38, 2004.

[8] Lin Chen, Richi Nayak, Sangeetha Kutty, and Yue Xu. Users segmentations for recommendation. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013.

[9] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[10] Souvik Debnath, Niloy Ganguly, and Pabitra Mitra. Feature weighting in content based recommendation system using social network analysis. In *Proceedings of the 17th international conference on World Wide Web*, pages 1041–1042. ACM, 2008.

[11] Alberto Díaz and Pablo Gervás. Personalisation in news delivery systems: Item summarization and multitier item selection using relevance feedback. *Web Intelligence and Agent Systems*, 3(3):135–154, 2005.

[12] Ulrich Drepper. What every programmer should know about memory. *Red Hat, Inc*, 11, 2007.

[13] D.M. Endres and J.E. Schindelin. A new metric for probability distributions. *Information Theory, IEEE Transactions on*, 49(7):1858–1860, 2003.

[14] A Go, L Huang, and R Bhayani. Twitter sentiment analysis. *Entropy*, 2009(June):17, 2009.

[15] J. Hannon, M. Bennett, and B. Smyth. Recommending twitter users to follow using content and collaborative filtering approaches. In *Proceedings of the 4th ACM conference on Recommender systems*, pages 199–206. ACM, 2010.

[16] John Hannon, Mike Bennett, and Barry Smyth. Recommending twitter users to follow using content and collaborative filtering approaches. In *Proceedings of the 4th ACM conference on Recommender systems*, RecSys '10, pages 199–206, New York, NY, USA, 2010. ACM.

[17] O. Hartig, C. Bizer, and J.C. Freytag. Executing sparql queries over the web of linked data. *The Semantic Web-ISWC 2009*, pages 293–309, 2009.

[18] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, January 2004.

[19] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, WebKDD/SNA-KDD '07, pages 56–65, NY, USA, 2007. ACM.

[20] Henry Kautz, Bart Selman, and Mehul Shah. Referral web: combining social networks and collaborative filtering. *Communications of the ACM*, 40(3):63–65, 1997.

[21] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009.

[22] J Lafferty, A McCallum, and F Pereira. *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[23] Pawan Lingras, Mofreh Hogo, and Miroslav Snorek. Interval set clustering of web users using modified kohonen self-organizing maps based on the properties of rough sets. *Web Intelligence and Agent Systems*, 2(3):217–225, 2004.

[24] Pasquale Lops, Marco Gemmis, and Giovanni Semeraro. Content-based Recommender Systems: State of the Art and Trends Recommender Systems Handbook. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, chapter 3, pages 73–105. Springer US, Boston, MA, 2011.

[25] Ching man Au Yeung, Nicholas Gibbins, and Nigel Shadbolt. A study of user profile generation from folksonomies. In Peter Dolog, Markus KrÃűtzsch, Sebastian Schaffert, and Denny Vrandecic, editors, *SWKM*, volume 356 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[26] Y. Matsuo and M. Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(1):157–170, 2004.

[27] Prem Melville, Raymod J. Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Eighteenth national conference on Artificial intelligence*, pages 187–192, Menlo Park, CA, USA, 2002. AAAI.

[28] Peter Mika. Flink: Semantic web technology for the extraction and analysis of social networks. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):211–223, 2005.

[29] Sergio Muñoz, Jorge Pérez, and Claudio Gutierrez. Minimal deductive systems for rdf. In *The Semantic Web: Research and Applications*, pages 53–67. Springer, 2007.

[30] A. Pak and P. Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. *Proceedings of LREC 2010*, 2010.

[31] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, 1(1):19–30, 2010.

[32] F. Peng and A. McCallum. Information extraction from research papers using conditional random fields. *Information processing & management*, 42(4):963–979, 2006.

[33] M. Pennacchiotti and S. Gurumurthy. Investigating topic models for social media user recommendation. In *Proceedings of the 20th international conference companion on World wide web*, pages 101–102. ACM, 2011.

[34] O. Phelan, K. McCarthy, and B. Smyth. Using twitter to recommend real-time topical news. In *Proceedings of the third ACM conference on Recommender systems*, pages 385–388. ACM, 2009.

[35] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[36] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. The adaptive web. chapter Collaborative filtering recommender systems, pages 291–324. Springer-Verlag, Berlin, Heidelberg, 2007.

[37] Y. Song, Z. Zhuang, H. Li, Q. Zhao, J. Li, W.C. Lee, and C.L. Giles. Real-time automatic tag recommendation. In *Proceedings of the 31st annual international*

ACM SIGIR conference on Research and development in information retrieval, pages 515–522. ACM, 2008.

[38] Johann Stan, Viet-Hung Do, and Pierre Maret. Semantic user interaction profiles for better people recommendation. In *ASONAM*, pages 434–437, 2011.

[39] Herman J ter Horst. Completeness, decidability and complexity of entailment for rdf schema and a semantic extension involving the owl vocabulary. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):79–115, 2005.

[40] Peter D. Turney and Patrick Pantel. From frequency to meaning: vector space models of semantics. *Journal of Artificial Intelligence Research*, 37(1):141–188, January 2010.

[41] Manolis Vozalis and Konstantinos G Margaritis. On the enhancement of collaborative filtering by demographic data. *Web Intelligence and Agent Systems*, 4(2):117–138, 2006.

[42] H.M. Wallach. Topic modeling: beyond bag-of-words. In *Proceedings of the 23rd international conference on Machine learning*, pages 977–984. ACM, 2006.

[43] Christo Wilson, Bryce Boe, Alessandra Sala, Krishna P.N. Puttaswamy, and Ben Y. Zhao. User interactions in social networks and their implications. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, pages 205–218, New York, NY, USA, 2009. ACM.

[44] W. Wu, B. Zhang, and M. Ostendorf. Automatic generation of personalized annotation tags for twitter users. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 689–692. Association for Computational Linguistics, 2010.

[45] E. Zangerle, W. Gassler, and G. Specht. Using tag recommendations to homogenize folksonomies in microblogging environments. *Social Informatics*, pages 113–126, 2011.

[46] T. Zesch and I. Gurevych. Approximate matching for evaluating keyphrase extraction. In *Proceedings of the 7th International Conference on Recent Advances in Natural Language Processing*, pages 484–489. Citeseer, 2009.