Algorithme top-k pour la recherche d'information dans les réseaux sociaux

Talel Abdessalem Bogdan Cautis
Silviu Maniu
Télécom ParisTech - CNRS LTCI
first.last@telecom-paristech.fr

Résumé

Nous nous intéressons dans cet article à la recherche d'information dans les réseaux sociaux, en particulier les réseaux de partage de marque-pages Internet (social tagging systems) comme delicious. Nous considérons que les requêtes sont formulées par des membres du réseau, et qu'elles portent sur des ressources partagées et classées selon le principe de la folksonomie, par des mots clés (tags). Etant donné, la taille gigantesque de certains réseaux, il est nécessaire de proposer des solutions efficaces, rapides et qui passent à l'échelle. La solution proposée dans cet article exploite les liens entre les membres du réseau pour déduire le degré de proximité entre l'auteur d'une requête et les autres membres du réseau. Ces poids servent à guider l'algorithme de recherche d'information vers les ressources supposées être les plus pertinentes pour l'auteur de la requête. Ceci est fait en ligne et sans calculs préalables. Les tests menés sur des jeux de données réels montrent l'efficacité de notre algorithme.

Mots clés : top-k algorithms, information retrieval, social network, folksonomies

1 Introduction

Unprecedented volumes of data are now at everyone's fingertips on the World Wide Web. The ability to query them efficiently and effectively, by performant

retrieval and ranking algorithms, has largely contributed to the rapid growth of the Web, making it simply irreplaceable in our every day life.

A new dynamics to this development has been recently brought by the *social Web*, applications that are centered around users, their relationships and their data. Indeed, user-generated content is becoming a significant and highly qualitative portion of the Web. To illustrate, the most visited Web site today is a social one. This calls for adapted, efficient retrieval techniques, which can go beyond a classic Web search paradigm where data is decoupled from the users querying it.

An important class of social applications are the *collaborative tagging sites*, also known as *folksonomies*, with popular examples including Del.icio.us, StumbleUpon or Flickr. Their general setting is the following :

- users form a *social network*, which may reflect proximity, similarity, friendship, closeness, etc,
- items (e.g., document, URLs, photos, etc) are *tagged* by users with keywords, for purposes such as description and classification, or to facilitate later retrieval,
- users *search* for items having certain keywords (i.e., tags) or they are *recommended* items, e.g., based on proximity at the level of tags.

Folksonomies, and social applications in general, can offer an entirely new perspective to how one searches and accesses information. The main reason for this is that users can (and often) play a role at both ends of the information flow, as producers and also as seekers of information. Consequently, finding the most relevant items that are tagged by some keywords should be done in a *network-aware* manner. In particular, items that are tagged by the users who are closer to the seeker should be given more weight than the items that are tagged by more distant users.

We consider in this paper the problem of top-k retrieval in collaborative tagging systems. We investigate this problem with a focus on efficiency, targeting techniques that have the potential to scale to current applications on the Web. (Note that the most popular ones have user bases of the order of millions and huge repositories of data ; today's most accessed social Web application, which also provides tagging and searching functionalities, has more than half a billion registered users.)

We associate with the notion of social network a rather general interpretation, as a user graph whose edges are labeled by *social scores*, which give a measure of the proximity or similarity between two users. These are then exploitable in searches, as they say how much weight one's tagging actions should have in the result build-up. For example, even for tagging applications where an explicit social network is not present or is not exploitable, one may use the tagging history

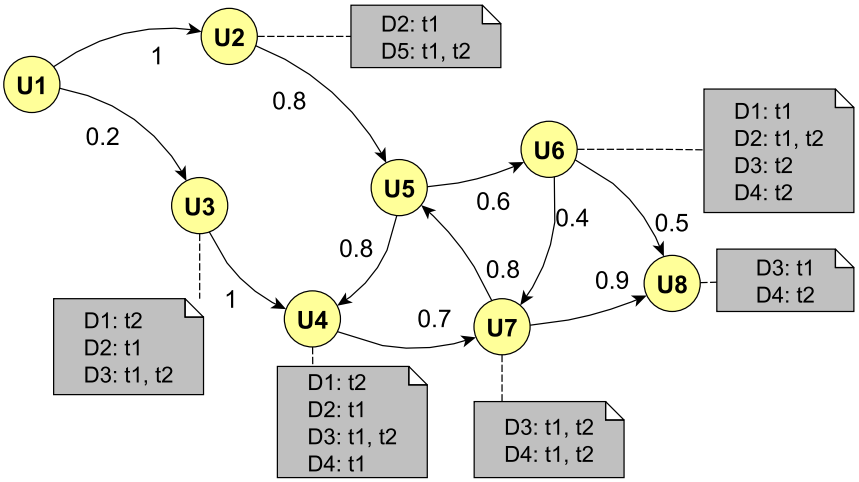

FIGURE 1 – A folksonomy and its social network.

of users to build a network based on similarity in tagging and items of interest. While we focus mainly on folksonomies, we believe that these represent a good abstraction for other types of social applications, to which our techniques could directly apply.

The core problem for top-k retrieval raised in our setting is the following : computing scores of top-k candidate items by iterating not only through the most relevant documents with respect to the query, but also (or mostly) by looking at the closest users and their tagged items (the term "closest" depends on certain model assumptions and will be clarified shortly).

Example 1 *Consider the collaborative tagging configuration of Figure 1. Users have associated lists of tagged documents and they are interconnected by social links. Each link is labeled by its (social) score, assumed to be in the* $[0, 1]$ *interval. Let us consider user* $u1$ *in the role of the seeker. The user graph is not complete, as the figure shows, and only two users have an explicit social score with respect to* $u1$*. For the remaining ones,* $u4, \ldots, u8$*, only an implicit social score could be computed from the existing links if a precise measure of their relevance with respect to* $u1$*'s queries is necessary in the top-k retrieval.*

Let us assume that $u1$ *looks for the top* 3 *documents that are tagged with both* $t1$ *and* $t2$*. Looking at* $u1$*'s immediate neighbors and their respective documents, intuitively,* $D5$ *should have a higher score than* $D3$*, given that the former is tagged*

by a more relevant user ($u1$, whose social score is the maximal one). If we expand the search to the entire social graph, the score of $D3$ may however benefit from the fact that other users, such as $u4$ or even $u8$, also tagged it with $t1$ or $t2$. Furthermore, documents such as $D4$ and $D2$ may also be relevant for the top 3 result, even though they were tagged only by users who are indirectly linked to $u1$.

Under certain assumptions to be clarified shortly, the top 3 documents for $u1$'s query will be, in descending score order, $D3$, $D2$ and $D4$. The rest of the paper will present the underlying model and the algorithm that allows us to compute this answer.

Classic top-k retrieval algorithms, such as Fagin's threshold algorithm [8] and the no random access (NRA) algorithm, rely on precomputed inverted-index lists with exact scores for each query term (in our setting, a term is a tag). Revisiting the setting in Figure 1, we would have two per-tag inverted lists $IL_{t1} = \{D3 : 4, D2 : 4, D4 : 2, D5 : 1, D1 : 1\}$ and $IL_{t2} = \{D3 : 4, D4 : 3, D1 : 2, D5 : 1, D2 : 1\}$, which give the number of times a document has been tagged with the given tag.

When user proximity is an additional ingredient in the top-k retrieval process, a direct adaptation of the threshold algorithm and variants in the network-aware setting would need precomputed inverted-index lists for each user-tag pair. For instance, if we interpret explicit links in the user graph as friendship, ignoring the link scores, and only tagging by direct friends matters, $u1$'s lists would be $IL_{u1,t1} = \{D2 : 2, D3 : 1, D5 : 1\}$ and $IL_{u1,t2} = \{D3 : 1, D1 : 1, D5 : 1\}$. Other 14 such lists would be required and, clearly, this would have prohibitive space and computing costs in a real-world setting. The work presented in [1] is the first to address this issue, considering the problem of network-aware search in collaborative tagging sites, though by a simplified flavor. More precisely, the authors consider an extension to classic top-k retrieval in which user proximity is seen as a binary function ($0-1$ proximity) : only users who are directly connected to the seeker can influence the top-k result. This introduces two strong simplifying restrictions : (i) only documents tagged by the seeker's friends should be relevant in the search, and (ii) all the friends of a given seeker are equally important.

The base solution of [1] is to keep for each tag-item pair, instead of the detailed lists per user-tag pair, only an upper-bound value on the number of taggers : the maximal number of taggers from any user's neighborhood. For instance, the upper-bound for $(t1, D2)$ would be 2, since for any user there are at most two neighbors who tagged $D2$ with $t1$. This is called the GLOBAL UPPER-BOUND strategy. A more refined version, which trades space for efficiency, keeps such upper-bound values within *clusters* of users, instead of the network as a whole.

The main drawback of [1]'s approach is that it cannot apply under a more general social search interpretation, in which a tagger's importance for the top-k result depends on how strongly she is related to the seeker (without necessarily being a direct friend thereof).

In [14], the network-aware retrieval problem for collaborative tagging is considered under a general interpretation, the one we also adopt in this paper. In considers that even users who are only indirectly connected to the seeker can be relevant for the top-k result. Their CONTEXTMERGE algorithm follows the intuition that the users closest to the seeker will contribute more to the score of an item, thus maximizing the chance that the item will remain in the final top-k. The authors describe a hybrid approach in which, at each step, the algorithm chooses either to look at the documents tagged by the closest unseen user or at the tag-document inverted-index lists (a seeker agnostic choice). In order to obtain the next (unseen) closest user at any given step, the algorithm precomputes in advance the proximity value for all possible pairs of users. These values are then stored in ranked lists (one list per user), and a simple pointer increment allows to obtain the next relevant user.

Example 2 *Consider the network of Figure 1. With respect to seeker* $u1$*, the list of users ranked by their proximity would be*

$$\{u2 : 1, u5 : 0.8, u4 : 0.64, u6 : 0.6, u7 : 0.44, u8 : 0.3, u3 : 0.2\},$$

when proximity for two users is computed by the maximal product of scores over paths linking them.

The main drawback of [14] is scalability. Clearly, precomputing a weighted transitive closure over the entire network has very high costs in terms of space and computation in even moderate-size social networks. More importantly, keeping these proximity lists up to date when they reflect tagging similarity [1] (as advocated in [14]), would simply be unfeasible in real-world settings, which are highly dynamic. (We revisit these considerations in Section 4.)

Main contributions. We propose an algorithm for top-k answering in collaborative tagging, which has the potential to scale to current applications and beyond. We first consider a key aspect of the problem : accessing efficiently the closest users for a given seeker. We describe how this can be done on the fly (without any

1. For top-k search by tags, tagging similarity may indeed be a more pertinent proximity measure than friendship.

pre-computations) for several possible choices of proximity computation in a social network, arguably the most natural ones. The interest in doing this is twofold :

- we can iterate over the relevant users in more efficient manner, since a typical network can easily fit in main-memory ; this can spare the potentially huge disk volumes required by [14]'s algorithm (see Section 4), while also having the potential to run faster.
- when the social network depends on the tagging history, we can keep up-to-date the network and, by it, all the proximity values at any given moment, with very little overhead.

Based on this, our top-k algorithm is sound and complete, and exhibits the same behavior as the one from previous literature. For further efficiency, we analyze in more depth this behavior on a Del.icio.us dataset and identify promising directions for settings where trading completeness is acceptable.

Other related work. The topic of search in a social setting has received increased attention lately. Studies and models of personalization of social tagging sites can be found in [15, 9, 7, 16]. Other studies have found that including social knowledge in scoring models can improve search and recommendation algorithms. In [5], personalization based on a similarity network is shown to outperform other personalization approaches and the non-personalized social search. A study on a last.fm dataset in [10] has found that incorporating social knowledge in a graph model system improves the retrieval recall of music track recommendation algorithms. The scoring model used in [14] is revisited in [17]. There, a textual relevance and a social influence score are combined in the overall scoring of items, the latter being calculated as the inverse of the shortest path between the seeker and the document publishers. An architecture for social data management is given in [2, 3], along with a framework for information discovery and presentation in social content sites.

The rest of the paper is organized as follows. In Section 2 we formalize the top-k retrieval problem in collaborative tagging applications. We describe a key aspect of our approach, the on-the-fly computation of proximity in Section 2.1. We then describe our top-k algorithm, in an exclusively social form, in Section 3 (the description of the algorithm for the general case can be found in the technical report). We discuss performance and scalability issues in Section 4. We conclude by providing in Section 5 some insight on the behavior of the algorithm on a Del.ico.us dataset, which motivates potential directions for efficiency by approximation.

2 General setting

We consider a social setting in which we have a set of items (these could be text documents, URL, photos, etc)

$$\mathcal{I} = \{i_1, i_2, \ldots, i_m\},$$

each tagged with one or more distinctive tags from a dictionary of tags $\mathcal{T} = \{t_1, t_2, \ldots, t_l\}$ by one or more users from $\mathcal{U} = \{u_1, \ldots, u_n\}$. We assume that the users form an undirected weighted graph $G = (\mathcal{U}, E, \sigma)$ called the *social network*. In G, each node represents a user and σ is a function that associates to each edge $e = (u_1, u_2)$ a value in $(0, 1]$, called *the proximity* (or social score) between users u_1 and u_2.

Given a seeker user s, a keyword query $Q = (t_1, ..., t_r)$ (composed of a set of r distinct tags) and an integer value k, the top-k retrieval problem is to compute the (possibly ranked) list of the k items having the highest scores with respect to the seeker and query.

Extending the model for social tagging systems presented in [1], we also assume the following two relations for tags :

- **tagging** : $Tagged(v, i, t)$: says that a user v tagged the item i with tag t,
- **tag proximity** : $SimTag(t_1, t_2, \lambda)$: says that tags t_1 and t_2 are similar, with similarity value $\lambda \in (0, 1)$.

We assume that a user can tag a given item with a given tag at most once. We first model for a user, item and tag triple (u, i, t) the *score* of item i for the given seeker u and tag t. This is denoted $score(i \mid u, t)$. Throughout this paper we use a "social" scoring function similar to the one of [14], which amounts to a simplified form of $BM25$, as follows :

$$score(i \mid u, t) = \frac{(p+1)fr(i \mid u, t)}{p + fr(i \mid u, t)} \times idf(t), \tag{2.1}$$

where $fr(i \mid u, t)$ is the *overall term frequency* of item i given the seeker u and tag t, $idf(t)$ is the *inverse document frequency* for tag t, and p is an application dependent parameter.

The inverse frequency is defined in fairly standard manner as follows :

$$idf(t) = \log \frac{|\mathcal{I}| - |\{i \mid Tagged(v, i, t)\}| + 0.5}{|\{i \mid Tagged(v, i, t)\}| + 0.5}. \tag{2.2}$$

The overall term frequency function $fr(i \mid u, t)$ is defined as a combination of a network-dependent component and a document-dependent one, as follows :

$$fr(i \mid u, t) = \alpha \times tf(t, i) + (1 - \alpha) \times sf(i \mid u, t). \tag{2.3}$$

The former component, $tf(t, i)$, is the term frequency of t in i, i.e., the number of times i was tagged with t. The latter component stands for social frequency, a measure that depends on the seeker.

If we consider that each user brings her own weight (proximity) to the score of an item, we can define the measure of social frequency as follows :

$$sf(i \mid u, t) = \sum_{v \in \{v \mid Tagged(v,i,t))\}} \sigma(u, v). \tag{2.4}$$

Alternatively, we might assume that only the most relevant user with respect to the seeker should determine the social frequency score of an item, thus having the following possible definition :

$$sf(i \mid u, t) = tf_t(i) \times max_{v \in \{v \mid Tagged(v,i,t)\}} \sigma(u, v) \tag{2.5}$$

Then, given a query q as a set of tags $(t_1, \ldots, t_r)$, the overall score of i for seeker u and query q,

$$score(i \mid u, q) = g(score(i \mid u, t_1), \ldots, score(i \mid u, t_r)),$$

is obtained using a monotone aggregate function g over the individual scores for each tag. In this paper, the aggregation function g is assumed to be a summation, $g = \sum_{t_j \in Q} score_{t_j}(i, u)$.

Extended proximity. The above scoring model takes into account only the neighborhood of the seeker (the users directly connected to her). But this can be extended to deal also with users that are indirectly connected to the seeker (friends-of-friends and beyond). We denote by σ^+ an *extended proximity*, which should be computable from σ for any pair of users connected by a path in the network. Now, σ^+ can replace σ in the two definitions of social frequency we consider before (Equations (2.4) and (2.5)), yielding an overall item scoring scheme that depends on the entire network instead of only the seeker's vicinity. We discuss shortly possible alternatives for σ^+ by means of aggregating σ values along paths in the graph. In the rest of this paper, when we talk about proximity we refer to the extended one.

For a given seeker u, by her *proximity vector* we denote the list of users with non-zero proximity with respect to u, ordered in descending order of these proximity values.

Remark 1. In Equation (2.3), the α parameter allows to tune the relative importance of the social component with respect to classic term frequency. When α is valued 1, the score becomes network-independent. On the other hand, when α is valued 0 the score depends exclusively on the social network. We will describe in this paper only the top-k retrieval algorithm for this particular case, its generalization to arbitrary α values is given in the technical report.

Remark 2. Note that a network in which all the user pairs have a proximity score of 1 amounts to the classical document retrieval setting (i.e., the result is independent of the user asking the query).

Remark 3. Tag similarity can be integrated into Equation 2.4, e.g., , by setting a threshold τ such that when $SimTag(t, t', \lambda)$, with λ above τ, and $Tagged(v, i, t')$, we also add $\sigma(u, v)$ to
$sf(i \mid u, t)$. For the sake of simplicity this is ignored in the description of our approach, but remains an integral part of the model.

2.1 Possible definitions for σ^+

We describe in this section a key aspect of our algorithm for top-k search, namely on-the-fly computation of proximity values with respect to a seeker s. The issue here is to facilitate at any given step the retrieval of the most relevant unseen user u in the network, along with her proximity value $\sigma^+(s, u)$. This user will have the potential to contribute the most to the partial scores of items that are still candidates for the top-k result, by Equations (2.4) and (3.1).

We start by considering possible definitions for σ^+, drawing inspiration from studies in the area of trust propagation for belief statements.

Candidate 1. Experiments on trust propagation in the Epinions network (for computing a final belief in a statement) [13] or in P2P networks show that (i) multiplying the weights on a given path between u and v, and (ii) choosing the maximum value over all the possible paths, gives the best results (measured in terms of precision and recall) for predicting beliefs. We can adapt this to our scenario by assuming that belief refers to *tagging with a tag* t. We thus aggregate the weights on a path $p = (u_1, \ldots, u_l)$ (with a slight abuse of notation) as

$$\sigma^+(p) = \prod_i \sigma(u_i, u_{i+1}).$$

For seeker $u1$ in our running example, we gave in the previous section the proximity values and the ordering of the network under this candidate for σ^+.

Candidate 2. A possible drawback of Candidate 1 for proximity aggregation is that values may decrease quite rapidly by multiplication. A σ^+ function that avoids this could be obtained by replacing multiplication over a path with minimal, as follows :

$$\sigma^+(p) = min\{\sigma(u_i, u_{i+1})\}.$$

Under this candidate for σ^+, the values with respect to seeker $u1$ would be the following :

$$\{u2 : 1, u5 : 0.8, u4 : 0.8, u7 : 0.7, u8 : 0.7, u6 : 0.6, u3 : 0.2\}.$$

Candidate 3. Another possible definition for σ^+ we consider relies on an aggregation over a path that penalizes long paths, i.e., distant users, as follows :

$$\sigma^+(p) = 2^{-\sum_i \frac{1}{\sigma(u_i, u_{i+1})}}.$$

Under this candidate for σ^+, the rounded values with respect to seeker $u1$ would be the following :

$$\{u2 : 0.5, u5 : 0.21, u4 : 0.08, u6 : 0.06, u7 : 0.03, u3 : 0.03, u8 : 0.01\}.$$

Using any of the three candidate definitions for weight aggregation over one path, we then define σ^+ for any pair of user who are connected in the network by taking the maximal weight over all their connecting paths. More formally, we define $\sigma^+(s, u)$ as

$$\sigma^+(s, u) = max_p\{\sigma^+(p) \mid s \overset{p}{\rightsquigarrow} u\}. \tag{2.6}$$

Note that when the first candidate (multiplication) is used, we reach the same aggregation scheme as in [13], which is also the one employed in [14] in the context of top-k network aware search.

Example 3 *For our running example, if we use Candidate 1 in Equation (2.6) and we assume that the seeker is* $u1$*, for* $\alpha = 0$ *(hence exclusively social relevance), by Equation 2.3 we obtain the following values for social frequency :* $SF_{(u1,t1)} = \{D2 : 2.44, D3 : 1.58, D4 : 1.08, D5 : 1, D1 : 0.6\}$ *and* $SF_{(u1,t2)} = \{D3 : 1.88, D4 : 1.1, D5 : 1, D1 : 0.84, D2 : 0.6\}$.

We argue next that to all these possible flavors for computing user proximity a greedy approach is applicable. This will allow us to browse the network of users

on the fly, at query time, visiting them in the order of their proximity with respect to the seeker. More precisely, in style similar to Dijkstra's algorithm [6], we will maintain a max-priority queue whose top element will be at any moment the *most relevant unvisited user*[2]. A user is *visited* when her tagged items are taken into account in the top-k result computation, as described in the following sections (this can occur at most once).

At each step advancing in the network, the top of the queue is extracted (visited) and its unvisited neighbours (adjacent nodes) are added to the queue (if not already present) and are *relaxed* . Let $\odot$ denote the aggregation function over a path (one among the three candidates). Relaxation updates the best proximity score of these nodes, as follows :

Algorithm 1 Relaxation

if $\sigma^+(s,u) \odot \sigma(u,v) > \sigma^+(s,v)$ **then**
  $\sigma^+(s,v) = \sigma^+(s,u) \odot \sigma(u,v))$
  $v.previous = u$
end if

We can prove that this greedy approach allows us to visit the nodes of the network in decreasing order of their proximity with respect to the seeker. This is because the following property holds for all three candidates for σ^+ :

Property 1 *Given a social network G and a seeker user s, for any other user v in G that is connected to s we have*

$$\sigma^+(s,v) \geq \sigma^+(s, v.previous).$$

We describe in the following section how this greedy procedure for iterating over the network is exploited in our top-k social retrieval algorithm.

3 Top-k algorithm for $\alpha = 0$

As the focus of this paper is on the "social" aspects of our technique, we detail here our top-k algorithm for the special case when the parameter α is 0 (its extension the arbitrary α values is given in a companion technical report). As

2. Dijkstra's classic algorithm [6] computes single-source shortest paths in a weighted graph without negative edges.

usual, we assume that for each tag t there is an inverted-index list giving the items i tagged by it, in descending order of the term frequencies $tf(t, i)$. Also, for each user u and tag t, there is a precomputed projection over the *Tagged* relation for them, giving the list of items tagged by u with t (no particular order is assumed).

For $\alpha = 0$, the term frequency $tf(t, i)$ of an item is no longer relevant : how often an item i was tagged with a given tag t does not matter, it only matters which users tagged i with t. From Equation (3.1), the score for one tag now becomes the following :

$$score(i \mid u, t) = \frac{(p+1)sf(i \mid u, t)}{p + sf(i \mid u, t)} \times idf(t). \tag{3.1}$$

We mainly detail the computation of social frequency, $sf(i \mid u, t)$, as it is the key parameter in the ranking of items. Since by $\alpha = 0$ we do not use metrics that are tag-only dependent, it is no longer necessary to treat each tag of the query as a distinct dimension and to visit each in round-robin style (as done in the threshold algorithm or in CONTEXTMERGE). We can simply get at each step, for the currently visited user, all the items that were tagged by her with query terms. We call this approach "user-at-a-time".

For each $t_j \in Q$, by $max\_users(i, t_j)$ we denote the maximal number of yet unvisited users who may have tagged item i with tag t_j. This value is initially set to the maximal term frequency of t_j over all items, $max\_tf(t_j)$, value which is available at the head of the inverted-index list of t_j.

Each time we visit a user u who tagged item i with t_j we can update $sf(i \mid s, t_j)$ by adding $\sigma^+(s, u)$ to it and we can remove 1 from $max\_users(i, t_j)$. When the latter reaches 0, the score of i w.r.t. t_j, $sf(i \mid s, t_j)$, will be final. This also gives us a possible termination condition, as discussed in the following.

At any moment in the process, the *optimistic* overall score for an item i that has been already seen, MAX_SCORE(i, q), will be estimated using as social frequency for each tag t_j of the query the following value :

$$sf(i \mid s, t_j) + top(H) \times max\_users(i, t_j),$$

while the *pessimistic* overall score MIN_SCORE(i, q), is estimated by the assumption that the current social frequencies, $sf(i \mid s, t_j)$, will be the final ones. For the top-k candidate items, we keep a list D of all items, sorted in descending order by their minimal possible scores, MIN_SCORE(i, q), as given before.

An upper-bound score of the yet unseen documents, MAX_SCORE_UNSEEN, can be estimated as follows

$$\text{MAX\_SCORE\_UNSEEN} = top(H) \times max_{tf}(t_j).$$

When both MAX_SCORE_UNSEEN and the maximal optimistic score of items that are already in D but not in its top-k are less than the pessimistic score of the last element in the current top-k of D (denoted D[k]), then the run of the algorithm can terminate, as we are guaranteed that the top-k can no longer change. (Note however that at this point the top-k items may have only partial scores. If a ranked answer is needed then the process of visiting users should continue.)

Algorithm 2 Top-k algorithm for $\alpha = 0$

Require: seeker s, query $Q = (t_1, \ldots, t_r)$
1: **for all** nodes v, tags t_j, items i **do**
2: $\sigma^+(s, v) = -\infty$
3: $sf(i \mid s, q_j) = 0$
4: **end for**
5: $\sigma^+(s, s) = 0$; $D = \emptyset$
6: $H \leftarrow$ max-priority queue of nodes u (sorted by $\sigma^+(s, u)$)
7: **while** $H \neq \emptyset$ **do**
8: u=EXTRACT_MAX(H) ;
9: **for all** i and t_j s.t $Tagged(u, i, t_j)$ **do**
10: $sf(i \mid s, t_j) \leftarrow sf(i \mid s, t_j) + \sigma^+(s, u)$
11: **if** $i \in D$ **then**
12: $max\_users(i, t_j) \leftarrow max\_users(i, t_j) - 1$
13: **else**
14: add i to D
15: $max\_users(i, t_j) \leftarrow max\_tf(t_j) - 1$
16: **end if**
17: **end for**
18: **for all** users v s.t. $\sigma(u, v) \in E$ **do**
19: RELAX(u,v)
20: **end for**
21: **if** MIN_SCORE$(D[k], q) > max_{l>k}$(MAX_SCORE$(D[l], q)$ AND MIN_SCORE$(D[k], q) >$ MAX_SCORE_UNSEEN **then**
22: **break**
23: **end if**
24: **end while**
25: **return** $D[1], \ldots, D[k]$

We can prove the following property :

Property 2 *Given a social network G and a seeker user s, Algorithm 2 visits the users of the network in decreasing order of their σ^+ values with respect to s.*

As a corollary of the above property we have that Algorithm 2 visit the users who may be relevant for a query in the same order as the CONTEXTMERGE algorithm of [14] in the case when $\alpha = 0$.

4 Scaling and performance

We argue in this section that, in a real-world setting, our algorithm outperforms the one from existing literature both in terms of memory requirements and execution time.

Let us consider, as an illustrating example, one of the most popular folksonomies, Del.icio.us, which currently has probably around 10^7 users. (Twice less than that was reported in 2008 for Del.icio.us ; the image tagging system Flickr has a user base of comparable size.) Unsurprisingly, this social network is quite sparse, with an average degree of about 100. If a similar graph configuration would be maintained when weights (the σ function) are associated to the edges of the network (e.g., based on tagging proximity or some other measure) the size of an index that would precompute the extended proximity value for each pair of connected users in the network (the σ^+ function) would be roughly of 700 terabytes (i.e., $(10^7)^2 \times 7$ bytes, considering that 3 bytes are necessary for an user Id and 4 bytes are necessary for the float value of proximity). On the other hand, the weighted graph alone would require memory space of roughly 7 gigabytes (as $10^7 \times 100 \times 7$ bytes), and could easily fit in the RAM space of an average commodity workstation. Furthermore, existing techniques for network compression [4] might allow us to reduce the space required to store the network by a factor of $10 - 15$ while still supporting efficient updates and random access on compressed data.

The difference in memory requirements for the two alternatives becomes much more drastic when assuming a user base of the order of Facebook's social network, which currently consists of roughly 5×10^8 users (and is still growing at a fast pace). Precomputed lists for extended proximity go up to about 400 petabytes of memory space, while the network itself requires only about 400 gigabytes. The space needed to store the network can further decrease to fit RAM capacity that moderate commodity servers can provide today, if considering the compression techniques mentioned previously.

We next discuss the time performance aspects, which are however less important in our view, compared with the memory and updatability advantages that our algorithm presents.

Let n denote the number of users and let e denote the number of edges in the

	disk RA	disk SA	RAM access
CONTEXTMERGE	1	n	$(\|Q\| - 1) \times n$
Our algorithm	0	0	$O(n \lg n + e) + (\|Q\| - 1) \times n + n + e$

TABLE 1 – Comparison of computational costs for processing a query Q.

network. We assume without loss of generality that the query consists of a single tag (when the query has multiple tags, all dimensions can share the results of a single σ^+ computation).

For our algorithm, let us assume that the social network resides in main memory, e.g., by means of adjacency lists : for each vertex, we have a list of its neighbours and their associated weights (we can safely assume the list comes presorted descending by weight). For one top-k query execution, we will need at most $n + e$ operations to visit the entire network (we are guaranteed to take each vertex only once). For the proximity computation we can use a Fibonacci-heap based max-priority queue, since our graph is likely to be very sparse [12]. Each insertion into the heap takes $O(1)$ amortized time, each extraction takes $O(\lg n)$ and each increase of a key (a relaxation step) takes $O(\lg n)$, for an overall queue complexity of $O(n \lg n + e)$.

CONTEXTMERGE requires no computations for proximity at query time. However, it uses disk accesses to read the precomputed proximity values : one random access to locate the seeker's list and n sequential disk accesses to read this list. (It suffices to do this just for one query term, and then keep and access a shared copy of this list in main memory.)

If we value the time taken by a memory access as 1 and the time taken by a sequential disk access as t (usually about five orders of magnitude slower than RAM access), with minor simplifications, we have that our algorithm has the potential to perform better than CONTEXTMERGE when the following holds :

$$t > \lg n + \frac{e}{n}.$$

Put otherwise, the sparseness of the network should be as follows :

$$e < n \times (t - \lg n).$$

A summary of this comparison on execution time is presented in Table 1. Note that in this analysis we omitted initialization costs : the overhead necessary for CONTEXTMERGE to compute the σ^+ values for all possible user pairs and the overhead to load in main-memory the social network, for our algorithm.

5 Further improving efficiency

We conclude by giving in this section some considerations on improving efficiency when approximate top-k answers are accepted.

The algorithm described in the previous section is sound and complete, and requires no precomputed information on proximity. The downside to this is that, in practice, it may visit too many users and their documents before being able to conclude that the top-k answer can no longer change. Note that to test termination, we used the tightest possible conditions (which might indeed be met in a social tagging configuration), when no additional knowledge on the values of the proximity vector for a given seeker is available (this was also the assumption in [14]'s CONTEXTMERGE algorithm).

But if one has certain information on how the values in a proximity vector variate from the most relevant user to the least relevant one, this would enable us to use more refined termination conditions. Extensive experiments on Del.icio.us data show that in average the last top-k change occurs in general sooner, hence there is a clear opportunity to improve the running time by stopping the visit of the network earlier. Equally important, we found that proximity vectors (their sequences of values) can be tightly approximated by *power laws*. For more details on these experimental results we refer the reader to the technical report.

Hence one possible direction for reducing the number of visited users is to precompute and materialize for each seeker a power law approximation of its proximity vector. This would allow us to use a more accurate estimation for the remaining (unseen) users, instead of uniformly associating them the score of $top(H)$. Though this may obviously introduce approximations in the final result, it present two main advantages :

- negligible memory consumption : only a power law description for each seeker is required, so these could well fit in the RAM space,
- robustness in dynamic networks : even when the network is quite dynamic, and even when the social graph depends on tagging actions (e.g., reflects tagging similarity), the power law approximation of the proximity vector is much less subject to significant change under a reasonable number of updates. It would thus be sufficient to adjust these power law approximations only periodically.

Références

[1] Sihem Amer-Yahia, Michael Benedikt, Laks V. S. Lakshmanan, and Julia Stoyanovich. Efficient network aware search in collaborative tagging sites. In *VLDB*, 2008.

[2] Sihem Amer-Yahia, Jian Huang, and Cong Yu. Building community-centric information exploration applications on social content sites. In *SIGMOD*, 2009.

[3] Sihem Amer-Yahia, Laks V. S. Lakshmanan, and Cong Yu. Socialscope : Enabling information discovery on social content sites. In *CIDR*, 2009.

[4] Gregory Buehrer and Kumar Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *WSDM*, 2008.

[5] David Carmel, Naama Zwerdling, Ido Guy, Shila Ofek-Koifman, Nadav Har'el, Inbal Ronen, Erel Uziel, Sivan Yogev, and Sergey Chernov. Personalized social search based on the user's social network. In *CIKM*, 2009.

[6] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1959.

[7] Zhicheng Dou, Ruihua Song, and Ji-Rong Wen. A large-scale evaluation and analysis of personalized search strategies. In *WWW*, 2007.

[8] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[9] Paul Heymann, Georgia Koutrika, and Hector Garcia-Molina. Can social bookmarking improve web search ? In *WSDM*, 2010.

[10] Ioannis Konstas, Vassilios Stathopoulos, and Joemon M. Jose. On social networks and collaborative recommendation. In *SIGIR*, 2009.

[11] Silviu Maniu, Bogdan Cautis, and Talel Abdessalem. Efficient top-k retrieval in real social taggging networks. Technical Report. http ://perso.telecom-paristech.fr/ cautis/papers/socialTopK-draft.pdf.

[12] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *IMC*, 2007.

[13] Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. Trust management for the semantic web. In *ISWC*, 2003.

[14] Ralf Schenkel, Tom Crecelius, Mouna Kacimi, Sebastian Michel, Thomas Neumann, Josiane X. Parreira, and Gerhard Weikum. Efficient top-k querying over social-tagging networks. In *SIGIR*, 2008.

[15] Jun Wang, Maarten Clements, Jie Yang, Arjen P. de Vries, and Marcel J. T. Reinders. Personalization of tagging systems. *Inf. Process. Manage*, 2010.

[16] Shengliang Xu, Shenghua Bao, Ben Fei, Zhong Su, and Yong Yu. Exploring folksonomy for personalized search. In *SIGIR*, 2008.

[17] Peifeng Yin, Wang-Chien Lee, and Ken C.K. Lee. On top-k social web search. In *CIKM*, 2010.