Network-aware Search in Social Tagging Applications: Instance Optimality versus Efficiency

Silviu Maniu, Bogdan Cautis

Network-aware Search in Social Tagging Applications: Instance Optimality versus Efficiency *

Silviu Maniu
Department of Computer Science
University of Hong Kong
Pokfulam Road, Hong Kong
smaniu@cs.hku.hk

Bogdan Cautis
Université Paris-Sud &
INRIA Saclay
91405 Orsay Cedex, France
bogdan.cautis@u-psud.fr

ABSTRACT

We consider in this paper top-k query answering in social applications, with a focus on *social tagging*. This problem requires a significant departure from socially agnostic techniques. In a network-aware context, one can (and should) exploit the social links, which can indicate how users relate to the seeker and how much weight their tagging actions should have in the result build-up. We propose algorithms that have the potential to scale to current applications.

While the problem has already been considered in previous literature, this was done either under strong simplifying assumptions or under choices that cannot scale to even moderate-size real-world applications. We first revisit a key aspect of the problem, which is accessing the closest or most relevant users for a given seeker. We describe how this can be done on the fly (without any pre-computations) for several possible choices – arguably the most natural ones – of proximity computation in a user network. Based on this, our top-k algorithm is sound and complete, addressing the applicability issues of the existing ones. Moreover, it performs significantly better in general and is instance optimal in the case when the search relies exclusively on the social weight of tagging actions.

To further address the efficiency needs of online applications, for which the exact search, albeit optimal, may still be expensive, we then consider approximate algorithms. Specifically, these rely on concise statistics about the social network or on approximate shortest-paths computations. Extensive experiments on real-world data from Twitter show that our techniques can drastically improve response time, without sacrificing precision.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search Process*

Keywords

social applications; social search; threshold algorithms

*Work performed while the authors were affiliated with Institut Mines-Télécom - Télécom ParisTech.

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.

http://dx.doi.org/10.1145/2505515.2505760.

1. INTRODUCTION

Unprecedented volumes of data are now at everyone's fingertips on the Web. The ability to query them effectively, by fast retrieval and ranking algorithms, has largely contributed to the rapid growth of the Web, making it 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, one of the two most visited Web sites 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 *social tagging* ones, with popular examples including Delicious, Flickr, or Twitter. Their general setting is the following:

1. users form a *social network*, which may reflect proximity, similarity, friendship, closeness, etc,
2. items from a public pool of items (tweets, documents, URLs) are *tagged* by users with keywords (tags, hashtags), for description, classification, or to facilitate later retrieval,
3. users *search* for items having certain tags.

Social tagging, 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 do) 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 users who are "closer" to the seeker – where the term closer depends on model assumptions that will be clarified shortly – should be given more weight than items that are tagged by more distant users.

We consider in this paper the problem of top-k retrieval in social tagging systems, with a focus on efficiency, targeting techniques that have the potential to scale to current applications on the Web, in an online context where the network, the tagging data and even the seekers' search ingredients can change at any moment. In this context, a key sub-problem for top-k retrieval that we need to address is computing scores of top-k candidates by iterating not only through the most relevant items with respect to the query, but also (or mostly) by looking at the closest users and their tagged items.

We associate with the notion of social network a general interpretation: a user graph whose edges are labeled by *social scores*, which give a measure of proximity or similarity between users. While we focus on social tagging in this paper, we believe this represents a good abstraction for many types of social applications, to which our techniques could apply.

EXAMPLE 1. *Consider the social tagging configuration of Figure 1 (left). Users have associated lists of tagged documents and*

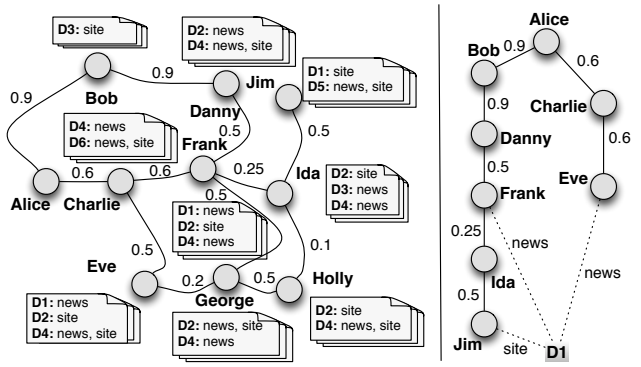

Figure 1: Social tagging scenario and its social network (left); some paths from a seeker towards a relevant item (right).

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 Alice in the role of the seeker. The user graph is not a complete one, as the figure shows, and only two users have an explicit social score with respect to Alice. For the remaining ones, Danny, . . . , Jim, only an implicit social score could be computed from the existing links if a precise measure of their relevance with respect to Alice's queries is necessary in the top-k retrieval.*

Let us assume that Alice looks for the top two documents that are tagged with $news$ *and* $site$*. Looking at Alice's immediate neighbors and their respective documents, intuitively,* $D3$ *should have a higher score than* $D4$*, since the former is tagged by a more relevant user (*Bob*, having the maximal social score relative to Alice). If we expand the search to the entire graph, e.g., via a shortest paths-like interpretation, the score of* $D4$ *may however benefit from the fact that other users, such as* Eve *or even* $Holly$*, also tagged it with* $news$ *or* $site$*. Furthermore, documents such as* $D2$ *and* $D1$ *may also be relevant for the top-*2 *result, even though they were tagged only by users who are indirectly linked to Alice.*

Figure 1 (right) gives a different perspective on our scenario, illustrating how Alice can reach one of the relevant documents, $D1$*, by following three paths in the social network. Intuitively, an aggregation of such paths from a seeker towards data items will be used in the scoring model. Under certain assumptions to be clarified shortly, the top-*2 *documents for Alice's query will be, in descending score order,* $D4$ *and* $D2$*. The rest of the paper will present the underlying model and algorithms that allow to build this answer.*

Main related work. Top-k retrieval algorithms, such as the Threshold Algorithm (TA) and the No Random Access algorithm (NRA) [12], 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 $IL(news) = \{D4 : 7, D2 : 2, D1 : 2, D3 : 1, D6 : 1, D5 : 1\}$ and $IL(site) = \{D2 : 5, D4 : 2, D3 : 1, D6 : 1, D1 : 1, D5 : 1\}$, 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 network-aware adaptation of the threshold algorithm and variants would need precomputed inverted list indices 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, Alice's lists would be $IL_{Alice}(news) = \{D4 : 1, D6 : 1\}$, $IL_{Alice}(site) = \{D3 : 1, D6 : 1\}$. Other 18 such lists would be required; clearly, this would have prohibitive space and computing costs in real settings.

Amer-Yahia et al. [1] is the first to address this issue, considering the problem of network-aware search in social tagging sites, though in a simplified setting. The authors consider an extension to classic top-k retrieval in which user proximity is seen as a binary function (0-1 proximity): only a subset of the users in the network are selected and can influence the top-k result. This introduces two strong simplifying restrictions: (i) only documents tagged by selected users should be relevant in the search, and (ii) all the users thus selected 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 on the number of taggers: the maximal number of taggers from any user's neighborhood. For example, the upper-bound for $(news, D4)$ would be 2, since for any user there are at most two neighbors who tagged $D4$ with $news$. A more refined version, which trades space for efficiency, keeps such upper-bounds for *clusters* of users, instead of the network as a whole.

Only in Schenkel et al. [19], the network-aware retrieval problem for social tagging is considered under a general interpretation, the one we also adopt in this paper. It 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 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. *In Fig. 1, for seeker Alice, the list of users ranked by proximity would be* $\{Bob : 0.9, Danny : 0.81, Charlie$*:*$0.6, Frank : 0.4, Eve : 0.3, George : 0.2, Holly : 0.1, Ida : 0.1, Jim : 0.05\}$*, with proximity between two users built as the maximal product of scores over paths linking them (formalized in Section 2.1). For example, the scores of* $Frank$ *and* Eve *correspond to paths of Fig. 1 (right), and will contribute to* $D1$*'s score w.r.t. the tag* $news$*.*

Our contributions. We present an algorithm for top-k answering in social tagging, which has the potential to scale to current online applications, where network changes and tagging are frequent. For it, we first address a key aspect: accessing efficiently the closest users for a given seeker. We describe how this can be done *on the fly* (without pre-computations) for a large family of functions for proximity computation in a social network, including the most natural ones (as the one used in [19]).

Based on this, our top-k algorithm SNSis sound and complete, and, when the search relies exclusively on the social weight of tagging actions, it is *instance optimal* – i.e., it visits a minimal number of users – in a large and important class of algorithms. Extensive experiments on real-world data from Twitter show that SNS performs significantly better than state-of-the-art techniques, up to two times faster (see Sec. 5). Going further, since in real-world online applications the joint exploration of the social and textual space may remain costly, even by optimal algorithms, we consider directions for approximate results. More specifically, these are motivated by the fact that estimates for termination conditions may be too loose in practice and that exact shortest paths-like computations may be too expensive in a large network. Our approaches present the advantages of negligible memory consumption – relying on concise statistics and pre-computed information about the social network and proximity values (via *landmarks*) – and reduced computation

overhead. Moreover, these statistics can be maintained up to date with little effort, even when the social network is built based on tagging history. Experiments show that approximate techniques can drastically improve the response time of the exact approach, even by an order of magnitude, with reduced impact on precision.

The main focus of our work is on the social aspects of top-k retrieval in tagging applications, and our techniques are designed to perform best in settings where tagging actions are mostly (if not exclusively) viewed through the lens of social relevance.

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 [20, 13, 11, 21]. Other studies have found that including social knowledge in scoring models can improve search and recommendation algorithms. In [8], 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 [16] has found that incorporating social knowledge in a graph model system improves the retrieval recall of music track recommendation algorithms. An architecture for social data management is given in [2, 3], with a framework for information discovery and presentation in social content sites. Another approach to rank resources in social tagging environments is CubeLSI [6], which uses a vector space model and extends Latent Semantic Indexing to include taggers in the feature space of resources, in order to better match queries to documents.

Several approaches for modifying the classic PageRank algorithm for bookmarking applications have been proposed. Algorithm FolkRank [14] proposes a ranking model in social bookmarking sites, for recommendation and search, based on an adaptation of PageRank over the tripartite graph of users, tags and resources. It follows the intuition that a resource that is tagged with important tags by important users becomes important itself and, symmetrically, for tags and users. An alternative approach to social-aware search, using personalized PageRank, was presented in [5]. There, the same tripartite model of annotators, resources and annotations is used to compute measures of similarities between resources and queries, and to capture the social popularity of resources. However, none of these approaches incorporate user-to-user relationships in their ranking model. In contrast, the social network is an integral part of the scoring model in our setting, if not the decisive one, while it can have various semantics (e.g., tagging similarity or trust).

The scoring model used in [19] is revisited in [22]. There, a textual relevance and a social influence score are combined in the overall scoring of items, the latter being computed as the inverse of the shortest path between the seeker and the document publishers. This model is also used in top-k retrieval of spatial web objects [7], where a prestige-based relevance score is computed by combining the overall relevance of an object with its spatial distance.

Person search and shortest paths related work gives another facet of "social search": the search of highly relevant persons for a given seeker and keyword query. Generally, the principle of approaches used in this type of application is to generate the k most relevant users, filtering them by the query keywords. These most relevant users are computed based on their shortest-path distances to the seeker, either in a centralized setting, as in [17], or in a distributed one, as in [4]. A characterization of proximity functions, including shortest paths, has also been recently given in [9]. These approaches are not directly applicable to our problem setting, as we have no prior knowledge on the identity and number of users who are highly relevant for the query and the top-k items to be returned. However, we describe here an adaptation to our setting of [17]'s landmark-based approach for shortest paths computations, in order to obtain faster approximate results.

Outline. We formalize in Sec. 2 the top-k retrieval problem in social tagging. We describe a key aspect of our approach, the on-the-fly computation of proximity in Sec. 2.1. We then describe our exact top-k algorithm, first in an exclusively social form, in Sec. 3, and show it is instance optimal (Sec. 3.1). The general algorithm is presented in Sec. 3.2. Approaches for improving efficiency by approximation are given in Sec. 4. Experimental results are presented in Sec. 5. and we discuss future research directions in Sec. 6.

2. NETWORK-AWARE TOP-K RETRIEVAL

In short, our model relies on scores that are obtained, either fully or partially, by some aggregation of shortest paths (in the social space) from a seeker towards good items (as illustrated in Figure 1, right).

More precisely, we consider a social setting in which we have a set of items (could be text documents, tweets, URLs, photos, etc) $\mathcal{I} = \{i_1, \ldots, i_m\}$, each tagged with one or more distinct tags from a dictionary $\mathcal{T} = \{t_1, t_2, \ldots, t_l\}$ by users from $\mathcal{U} = \{u_1, \ldots, u_n\}$. We assume that users form an undirected weighted graph $G = (\mathcal{U}, E, \sigma)$ called the *social network*. In G, nodes represent users and the σ function associates to each edge $e = (u_1, u_2)$ a value in $(0, 1]$, called *the proximity* (or social) score between u_1 and u_2.

Given a seeker user s, a keyword query $Q = (t_1, ..., t_r)$ (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.

We assume the following tagging relation $Tagged(v, i, t)$, which says that a user v tagged the item i with tag t (a user can tag a given item with a given tag at most once).

We first model by $score(i \mid s, t)$, for a seeker s, an item i and one tag t, the *score* of i for the given seeker and tag. Generally, $score(i \mid s, t) = h(fr(i \mid s, t))$, where $fr(i \mid s, t)$ is the *overall frequency* of item i w.r.t s and t, and h is a positive monotone function (e.g., tf-idf or BM25; see Section 5).

Overall frequency $fr(i \mid s, t)$ is defined as a combination of a network-dependent component and a document-dependent one:

$$fr(i \mid s, t) = \alpha \times tf(t, i) + (1 - \alpha) \times sf(i \mid s, t). \qquad (2.1)$$

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.[1]

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

$$sf(i \mid s, t) = \sum_{v \in \{v \mid Tagged(v,i,t))\}} \sigma(s, v). \qquad (2.2)$$

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

$score(i \mid s, Q) = g(score(i \mid s, t_1), \ldots, score(i \mid s, t_r))$,

is obtained using a monotone aggregate function g over the individual scores for each tag (in our experiments, the aggregation function g is assumed to be a summation, giving $\sum_{t_j \in Q} score(i \mid s, t_j)$).

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, following a natural interpretation that user links (e.g., similarity or trust) are (at least to some extent) transitive. We denote by σ^+ an *extended proximity*, which is

[1]The kind of linear combination of Eq. (2.1) is often used when a local retrieval score and a global one are to be combined, e.g., in spatial search [7] or in social search [19]; *any monotone combination* of the two score components can be used instead.

to be computable from σ for any pair of users connected by a path in the network. Now, σ^+ can replace σ in the definition of social frequency we consider before (Eq. (2.2)), 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.

2.1 Computing 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 Eq. (2.1) and (2.2).

We start by discussing possible candidates for σ^+, arguably the most natural ones, drawing inspiration from studies in the area of trust propagation for belief statements. We then give a wider characterization for the family of possible functions for proximity computation, to which these candidates belong.

Candidate 1(f_{mul}). Experiments on trust propagation in the Epinions network (for computing a final belief in a statement) [18] 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 integrate this into 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 $Alice$ in our running example, we gave in the previous section (Example 2) the proximity values and the ordering of the network under this candidate for σ^+.

Candidate 2(f_{min}). A possible drawback of Candidate 1 for proximity aggregation is that values may decrease quite rapidly. A σ^+ function that avoids this could be obtained by replacing multiplication over a path with minimal, as $\sigma^+(p) = \min_i\{\sigma(u_i, u_{i+1})\}$.

Under this σ^+ candidate, the values w.r.t. seeker $Alice$ would be $\{Bob : 0.9, Danny : 0.9, Charlie : 0.6, Frank : 0.6, Eve : 0.5, George : 0.5, Holly : 0.5, Ida : 0.25, Jim : 0.25\}$.

Candidate 3(f_{pow}). Another possible definition for σ^+ relies on an aggregation that penalizes long paths, i.e., distant users, in a controllable way, as $\sigma^+(p) = \lambda^{-\sum_i \frac{1}{\sigma(u_i,u_{i+1})}}$, where $\lambda \geq 1$ can be seen as an *exponential decay* factor; the greater its value the more rapid the decrease of proximity.[2] Under this candidate, for $\lambda = 2$, non-zero rounded values w.r.t $Alice$ are $\{Bob : 0.46, Charlie : 0.31, Danny : 0.21, Eve : 0.07, Frank : 0.052, George : 0.01\}$.

The key common feature of the candidate functions previously discussed is that they are monotonically non-increasing over any path, when σ draws values from the interval $[0, 1]$:

PROPERTY 1. *For a social network G and a path $p = \{u_1, \ldots, u_l\}$ in G, we have $\sigma^+(u_1, \ldots, u_l) \leq \sigma^+(u_1, \ldots, u_{l-1})$.*

We then define σ^+ for any pair of users (s,u) who are connected in the network by taking the maximal weight over all their connecting paths. More formally, we define $\sigma^+(s,u)$ as

$$\sigma^+(s,u) = max_p\{\sigma^+(p) \mid s \overset{p}{\rightsquigarrow} u\}. \quad (2.3)$$

[2]This is similar to Katz measures for social proximity [15].

Note that when the first candidate (multiplication) is used, we obtain the same aggregation scheme as in [18], which is also employed in [19] in the context of top-k network aware search.

EXAMPLE 3. *In our running example, if we use multiplication in Eq. (2.3), for the seeker Alice, for $\alpha = 0$ (hence exclusively social relevance), by Eq.(2.1) we obtain the following values for social frequency: $SF_{Alice}(news) = \{D4 : 2.6, D2 : 1.01, D1 : 0.7, D6 : 0.6, D3 : 0.1, D5 : 0.05\}$ and $SF_{Alice}(site) = \{D4 : 1.11, D2 : 1.1, D3 : 0.9, D6 : 0.6, D1 : 0.05, D5 : 0.05\}$.*

We argue next that to all aggregation definitions that satisfy Property 1 and apply Eq.(2.3), 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 w.r.t the seeker.

More precisely, by generalizing Dijkstra's shortest paths algorithm [10], we maintain a max-priority queue, denoted H, whose top element score $top(H)$ will correspond at any moment to the *most relevant unvisited user*. A user is *visited* when her tagged items are taken into account for the top-k result, 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 neighbors (adjacent nodes) are added to the queue (if not already present) and are *relaxed*. Let $\otimes$ denote the aggregation function over a path (satisfying Property 1).

It can be shown by straightforward induction that this greedy approach allows us to visit the nodes of the network in non-increasing order of their proximity with respect to the seeker, under any function for proximity aggregation that satisfies Property 1.

3. THE EXACT TOP-K ALGORITHM

As the main focus of this paper is on the social aspects of search in tagging systems, we detail first our top-k algorithm, SNS, for the special case when the parameter α is 0. In this case, $fr(i \mid s,t)$ is simplified as $fr(i \mid s,t) = sf(i \mid s,t)$.

For each user u and tag t, we assume a precomputed projection over the *Tagged* relation for them, giving the items tagged by u with t; we call these the *user lists*. No particular order is assumed for the items appearing in a user list. We keep a list D of candidate items, sorted in descending order by their *minimal possible scores* (to be defined shortly). An item becomes candidate when it is first met in a $Tagged$ triple.

As usual, we assume that, for each tag t, we have an inverted list $IL(t)$ giving the items i tagged by it, along with their term frequencies $tf(t,i)$[3], in descending order. The lists can be seen as unpersonalized indices; starting from the top item, they will be consumed one item at a time, whenever the current item becomes candidate for the top-k result. We denote by $CIL(t)$ the items already consumed (as known candidates), by $top\_item(t)$ the item present at the current (unconsumed) position of $IL(t)$, and we use $top\_tf(t)$ as short notation for the term frequency associated with this item.

We detail mostly the computation of social frequency, $sf(i \mid u,t)$, as it is the key parameter in the scoring function of items. Since when $\alpha = 0$ we do not use metrics that are tag-only dependent, it is not 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). It suffices for our purposes to get at each step, for the currently visited user, all the items that were tagged by her with query terms (one user list for each term).

[3]Even though the social frequency does not depend directly on tf scores, we will exploit the inverted lists and the tf scores by which they are ordered, to better estimate score bounds. In particular, as detailed later, this allows us to achieve instance optimality.

For each tag $t_j \in Q$, by $unseen\_users(i, t_j)$ we denote the maximal number of yet unvisited users who may have tagged item i with t_j. This is initially set to the maximal possible term frequency of t_j over all items (value that is available at the current position of the inverted list of $IL(t_j)$, as $top\_tf(t)$).

Each time we visit a user u who tagged item i with t_j we can (a) update $sf(i \mid s, t_j)$ (initially set to 0) by adding $\sigma^+(s, u)$ to it, and (b) decrement $unseen\_users(i, t_j)$.

When $unseen\_users(i, t_i)$ reaches 0, the social frequency value $sf(i \mid s, t_j)$ is final. This also gives us a possible termination condition, as discussed in the following.

At any moment in the run of the algorithm, the *optimistic* score MAXSCORE$(i \mid s, Q)$ of an item i that has already been seen in some user list is estimated using as social frequency, for each tag t_j of the query, the value $top(H) \times unseen\_users(i, t_j) + sf(i \mid s, t_j)$.

Symmetrically, the *pessimistic* overall score, MINSCORE$(i \mid s, Q)$, is estimated by the assumption that, for each tag t_j, the current social frequency $sf(i \mid s, t_j)$ will be the final one. The list of candidates D is sorted in descending order by this lowest possible score.

An upper-bound score on the yet unseen items, MAXSCOREUNSEEN is estimated using as social frequency for each tag t_j the value $top(H) \times top\_tf(t))$.

When the optimistic scores of items already in D but not in its top-k are less than the pessimistic score of the last item in the current top-k (i.e., $D[k]$), the algorithm can terminate; 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 and, if a ranked answer is needed, the process of visiting users should continue.)

Algorithm 1 gives the flow of SNS. Key differences w.r.t. CONTEXTMERGE's social branch are (i) the on-the-fly computation of proximity values, in lines 1-7 and 29-31, and (ii) the consuming of inverted list positions, when they become candidates, in lines 20-28. For clarity, we first exemplify a SNS run without the latter aspect (this would correspond to a CONTEXTMERGE run).

EXAMPLE 4. *Revisiting Example 1, recall we want the top-*2 *items for the query* $Q = \{news, site\}$ *from Alice's point of view. To simplify, let us assume that* $score(i \mid u, t) = sf(i \mid u, t)$ *and* g *is addition. We detail next the execution of our algorithm.*

At the first iteration of the line 8 loop, we visit Bob's user list, adding $D3$ *to the candidate buffer. At the second iteration, we visit* $Danny$*'s user list, adding* $D2$, $D4$ *to the buffer. At the third iteration (*$Carol$*'s user list) we add* $D6$ *to the buffer.* $D1$ *is added to the candidate buffer when the algorithm visits* $Frank$*'s user lists, at iteration 4. Recall* $top\_tf(news) = 7$ *and* $top\_tf(site) = 5$.

The 6th iteration of the algorithm is the final one, visiting $George$*'s user lists, finding* $D2$ *tagged with* $news$, $site$ *and* $D4$ *tagged with* $site$. $D4$ *and* $D2$ *are the top-*2 *candidates, with* MINSCORE$(D4, Q) = 2.61$ *and* MINSCORE$(D2, Q) = 2.21$. *The closest candidate is* $D6$, *with* MINSCORE$(D6, Q) = 1.2$ *and* MAXSCORE$(D6, Q) = 1.2 + 6 \times 0.1 + 4 \times 0.1 = 2.2$. *Also,* MAXSCOREUNSEEN$(Q) = 7 \times 0.1 + 5 \times 0.1 = 1.2$. *Finally,* MAXSCORE$(D6, Q) <$ MINSCORE$(D2, Q)$ *and since we have* MAXSCOREUNSEEN$(Q) <$ MINSCORE$(D2, Q)$, *the algorithm stops returning* $D4$ *and* $D2$.

We discuss next the interest of consuming the inverted list positions, when these become candidates (illustrated in Example 5). In lines 20-28, we aim at keeping to a minimum the worst-case estimation of the number of unseen taggers. More precisely, we test whether there are top-k candidates i (i.e., items already seen in user lists) for which the term frequency for some tag t_j of Q, $tf(t_j, i)$, is "within reach", as the one currently used (from $IL(t_j)$) as the basis for the optimistic (maximal) estimate of the number of yet unseen users who tagged candidate items with t_j. When such a pair (i, t_j) is found, we can do the following adjustments:

1. refine the number of unseen users who tagged i with t_j from a (possibly loose) estimate to its *exact* value; this is marked when i is added to the CIL list of t_j (line 24) and, from this point on, the number of unseen users will only change when new users who tagged i with t_j are found (line 18).
2. advance (one sequential access) beyond i in $IL(t_j)$ to the next best item, allowing to refine (at line 25) the estimates $unseen\_users(i', t_j)$ for all candidates i' for which the exact number of users who tagged with t_j is yet unknown.

Algorithm 1: SNS$_{\alpha=0}$: top-k algorithm for $\alpha = 0$

```
Require: seeker s, query Q = (t_1, ..., t_r)
 1: for all users u, tags t_j ∈ Q, items i do
 2:     σ+(s,u) = −∞
 3:     sf(i | s, t_j) = 0
 4:     set IL(t_j) position on first entry; CIL(t_j) = ∅
 5: end for
 6: σ+(s,s) = 0; D = ∅ (candidate items)
 7: H ← priority queue over nodes u, sorted by σ+(s,u)); initially {s}
 8: while H ≠ ∅ do
 9:     u=EXTRACT_MAX(H);
10:     for all tags t_j ∈ Q, triples Tagged(u,i,t_j) do
11:         sf(i | s, t_j) ← sf(i | s, t_j) + σ+(s,u)
12:         if i ∉ D then
13:             add i to D
14:             for all tags t_l ∈ Q do
15:                 unseen_users(i, t_l) ← top_tf(t_l)(initialization)
16:             end for
17:         end if
18:         unseen_users(i, t_j) ← unseen_users(i, t_j) − 1
19:     end for
20:     while ∃t_j ∈ Q s.t. i = top_item(t_j) ∈ D do
21:         tf(t_j, i) ← top_tf(t_j)(t_j's frequency in i is now known)
22:         advance IL(t_j) one position
23:         Δ ← tf(t_j, i) − top_tf(t_j) (the top_tf drop)
24:         add i to CIL(t_j)
25:         for all items i' ∈ D \ CIL(t_j) do
26:             unseen_users(i', t_j) ← unseen_users(i', t_j) − Δ
27:         end for
28:     end while
29:     for all users v s.t. σ(u,v) ∈ E do
30:         add v to H (if necessary) and RELAX(s,u,v)
31:     end for
32:     if MINSCORE(D[k], Q) > max_{l>k}(MAXSCORE(D[l], Q))
        AND MINSCORE(D[k], Q) > MAXSCOREUNSEEN then
33:         break
34:     end if
35: end while
36: return D[1], ..., D[k]
```

EXAMPLE 5. *Let us now consider how the choice of advancing in the inverted lists, when possible, influences the number of needed iterations. At first,* $top\_tf(news) = 7$, $top\_item(news) = D4$, *and* $top\_tf(site) = 5$, $top\_item(site) = D2$.

The first iteration only introduces $D3$ *and we cannot advance in any of the two inverted lists. Then, the discovery of* $D2$ *and* $D4$ *in step* 2 *allows us to fix their exact tf values and advance in the lists. The new positions are:* $top\_tf(news)$=2, $top\_item(news)$=$D1$, *and* $top\_tf(site)$=1, $top\_item(site)$=$D6$. $D6$*'s discovery in the* 3*rd iteration allows us to advance further in the inverted lists. Finally, in step* 4, *the discovery of* $D1$ *allows us to advance in the inverted lists to* $top\_tf(news) = 1$, $top\_item(news) = D5$, *and* $top\_tf(site) = 1$, $top\_item(site) = D5$ *(only undiscovered item). This allows for some drastic score estimation refinements. We have the same top-*2 *candidates,* $D4$ *and* $D2$, *with* MINSCORE$(D4, Q) = 1.81$, MINSCORE$(D2, Q) = 1.21$. *Closest item remains* $D6$, *by* MINSCORE$(D6, Q) =$ MAXSCORE$(D6, Q) = 1.2$, *since we know that we have visited all users who tagged*

$D6$. MAXSCOREUNSEEN$(Q) = 1 \times 0.3 + 1 \times 0.3 = 0.6$, *since the top unseen document could be tagged only once with each tag. Then, since* MAXSCOREUNSEEN$(Q) <$ MINSCORE$(D2, Q)$ *and* MAXSCORE$(D6, Q) <$ MINSCORE$(D2, Q)$ *we can exit the loop, two steps before the unrefined version, with* $D4$, $D2$ *as the top* 2.

We can prove the following property of our algorithm:

PROPERTY 2. *For a given seeker* s, $\text{SNS}_{\alpha=0}$ *visits the network in decreasing order of the* σ^+ *values with respect to* s.

As a corollary of Property 2, we have that $\text{SNS}_{\alpha=0}$ visits users who may be relevant for the query in the same order as the state-of-the-art algorithm of [19]. More importantly, we prove that our algorithm visits as few users as possible, i.e., it is instance optimal with respect to this aspect when $\alpha = 0$. Moreover, the experiments show that overall SNS can drastically reduce the number of visited user lists in practice (see Section 5).

3.1 Instance optimality of $\text{SNS}_{\alpha=0}$

We use the definition of instance optimality of [12]. For a class of algorithms **A**, a class of legal inputs (instances) **D**, $cost(\mathcal{A}, \mathcal{D})$ denotes the cost of running algorithm $\mathcal{A} \in \mathbf{A}$ on input $\mathcal{D} \in \mathbf{D}$. An algorithm $\mathcal{A}$ is *instance optimal* for its class **A** over inputs **D** if for every $\mathcal{B} \in \mathbf{A}$ and every $\mathcal{D} \in \mathbf{D}$ we have $cost(\mathcal{A}, \mathcal{D})$=$O(cost(\mathcal{B}, \mathcal{D}))$.

Let c_{UL} be the abstract cost of accessing the user list - a process which involves the relatively costly operations of finding the proximity value of the user and retrieving the items tagged by the user with query terms - and let $users(\mathcal{A}, \mathcal{D})$ be the number of user lists needed for establishing the top-k for algorithm $\mathcal{A}$ on input $\mathcal{D}$. Let c_S be the abstract cost of sequentially accessing the data in IL_t, and let $seqitems(\mathcal{A}, \mathcal{D})$ be the total number of sequential accesses to IL for algorithm $\mathcal{A}$ on input $\mathcal{D}$. In practice, $c_{UL} \gg c_S$ is a reasonable assumption, hence, for two algorithms $\mathcal{A}$ and $\mathcal{B}$, we have

$$\frac{users(\mathcal{A}, \mathcal{D}) \times c_{UL} + seqitems(\mathcal{A}, \mathcal{D}) \times c_S}{users(\mathcal{B}, \mathcal{D}) \times c_{UL} + seqitems(\mathcal{B}, \mathcal{D}) \times c_S} \approx \frac{users(\mathcal{A}, \mathcal{D})}{users(\mathcal{B}, \mathcal{D})}.$$

Hence, for a fair cost estimate in practical social search settings, a reasonable assumption is to consider $cost(\mathcal{A}, \mathcal{D}) = users(\mathcal{A}, \mathcal{D})$.

Let us now define the class of "social" algorithms **S** to which both $\text{SNS}_{\alpha=0}$ and CONTEXTMERGE (when $\alpha = 0$) belong. These algorithms correctly return the top-k items for a given query Q and seeker s, they do not use random accesses to $IL(t)$ indexes in order to fetch a certain tf value, and they do not include in their working buffers (e.g., candidate buffer D) items that were not yet encountered in the user lists. The last assumption could be seen as a "no wild guess" policy, by which the algorithm cannot guess that an item might be encountered in some later stages. This is a reasonable assumption in practice, as the number of items needed for computing a top-k result for a given seeker should in general be much smaller than the total number of items tagged by query terms.

The class **D** of accepted inputs consists of the inputs that respect the setting described in Section 2. We can prove the following result:

THEOREM 1. $\text{SNS}_{\alpha=0}$ *is instance optimal over* ***S*** *and* ***D****, when the cost is defined as* $cost(\mathcal{A}, \mathcal{D}) = users(\mathcal{A}, \mathcal{D})$.

3.2 Algorithm for the general case

For the general case of $\alpha \in [0, 1]$ – with the on-the-fly processing of user proximities – at each iteration, the algorithm can alternate, by calling CHOOSEBRANCH(), between two possible execution branches: the *social branch* (lines 8-31 of Algorithm 1) and the *textual branch*, which is a direct adaptation of the NRA algorithm.

As in the exclusively social setting of the previous section, we read tf-scores $tf(t_j, i)$ from the inverted lists, when needed, either as in line 21 of $\text{SNS}_{\alpha=0}$, or when advancing in the textual branch. Initially, all unknown tf-scores are assumed to be set to 0.

The optimistic overall score MAXSCORE(i, Q) of an item i that is already in the candidate list D will now be computed by setting $fr(i \mid s, t)$, defined in Eq. (2.1), to

$$fr(i \mid s, t) = (1 - \alpha) \times top(H) \times unseen\_users(i, t) + (1 - \alpha) \times sf(i \mid s, t) + \alpha \times max(tf(t, i), top\_tf(t)).$$

The last term accounts for the textual weight of the score, and uses either the exact term frequency (if known), or an upper-bound for it (the score in the current position of $IL(t)$).

Symmetrically, for the pessimistic overall score MINSCORE(i, Q), the frequency $fr(i \mid u, t)$ will be computed as

$$fr(i|s, t) = (1-\alpha) \times sf(i|s, t) + \alpha \times max(tf(t, i), partial\_tf(t)),$$

where $partial\_tf$ is the count of visited users who tagged i with t_j, and is used as lower-bound for $tf(t_j, i)$ when this is not yet known.

A score upper-bound for yet unseen items, MAXSCOREUNSEEN, is estimated using as overall frequency for each tag t_j the value

$$fr(i \mid s, t) = \alpha \times top\_tf(t) + (1 - \alpha) \times top(H) \times top\_tf(t)).$$

We present the flow of the general case algorithm in Algorithm 2. Method INITIALIZE() amounts to lines 1-6 of $\text{SNS}_{\alpha=0}$, and method PROCESSSOCIAL() amounts to lines 8-31 of $\text{SNS}_{\alpha=0}$ (modulo the straightforward adjustment for the count $partial\_tf$).

The difference between the $\alpha = 0$ case and the general case is the processing of the inverted lists (textual branch), which is done as in the No Random Access algorithm (NRA) [12] (see lines 7-13 of Algorithm 2). We discuss how the choice of the branch to be followed is done, by the CHOOSEBRANCH() subroutine hereafter.

Algorithm 2: SNS: top-k algorithm for the general case

```
Require: seeker s, query Q = (t_1, ..., t_r)
 1: INITIALIZE()
 2: while H ≠ ∅ do
 3:    CHOOSEBRANCH()
 4:    if social branch then
 5:       PROCESSSOCIAL()
 6:    else
 7:       for all tags t_j ∈ Q, item i = top_item(t_j) do
 8:          if i ∉ D then
 9:             add i to D and CIL(t_j)
10:          end if
11:          tf(t_j, i) ← top_tf(t_j)
12:          advance IL(t_j) one position
13:       end for
14:    end if
15:    if MINSCORE(D[k], Q) > max_{l>k}(MAXSCORE(d[l], Q) and
       MINSCORE(D[k], Q) > MAXSCOREUNSEEN then
16:       break
17:    end if
18: end while
19: return D[1], ..., D[k]
```

Choice between social and textual branches. $\text{SNS}_{\alpha=0}$ (when only the social branch matters) is instance optimal (Th. 1), with the cost being estimated as $users(\text{SNS}_{\alpha=0}, \mathcal{D})$. As the NRA algorithm, when only the textual branch matters, $\text{SNS}_{\alpha=1}$ is instance optimal, with the cost being estimated as $seqitems(\text{SNS}_{\alpha=0}, \mathcal{D})$.

When α is not one of the extreme values, under a cost function as a combination of the two above, of the form

$$users(\text{SNS}_{\alpha=0}, \mathcal{D}) \times c_{UL} + seqitems(\text{SNS}_{\alpha=1}, \mathcal{D}) \times c_S,$$

a key role for efficiency is played by CHOOSEBRANCH().

In [19], the choice between the textual branch or the social one was done by estimating the maximum potential score of each, in round-robin manner over the query dimensions. We use a different

– potentially more efficient – heuristic for the branch choice. At any point in the run of SNS, unless termination is reached, we have at least one item r with $\text{MAXSCORE}(r, Q) \geqslant \text{MINSCORE}(D[k], Q)$. We consider the item $r = D[argmax_{l>k}(\text{MAXSCORE}(D[l], Q)]$, which has the highest potential score, and we choose the branch that is the most likely to refine r's score (put otherwise, the branch that counts the most in the MAXSCORE estimation for r). The intuition behind this branch choice mechanism is that it is more likely to advance the run of the algorithm closer to termination.

For each tag $t_j \in Q$, we set $\text{MAXTEXTUAL}(t_j)$ to $\alpha \times top_t f(t_j)$ if the term frequency $tf(t_j, r)$ is not yet known, or to 0 otherwise. For the social part of the score, we set

$\text{MAXSOCIAL}(t_j) = (1 - \alpha) \times unseen\_users(t_j, r) \times top(H)$.

Then, we follow the social branch if, for at least one of the tags, MAXSOCIAL is greater than MAXTEXTUAL. Note that we deal with the tags of the query "in bulk", and advance simultaneously on their inverted lists when the textual branch is followed.

4. EFFICIENCY BY APPROXIMATION

The algorithm described in Section 3 is sound and complete, and requires no prior (aggregated) knowledge on the proximity values with respect to a certain seeker (e.g., statistics); this was also the assumption in [19]'s CONTEXTMERGE algorithm. Moreover, it is instance optimal in the exclusively social setting (one main focus of this paper) with respect to the number of visited users. While we improve the running time in both this setting and the general one (more on experiments in Section 5), in practice, however, two aspects can have a significant impact on efficiency:

Reason 1. The search may still visit a significant part of the network and users' item lists, before being able to conclude that the top-k answer is final. Yet we observed that, in most practical cases, the last top-k change occurs much sooner, so there is a clear opportunity to stop the browsing of the network earlier. For that, we can adopt tighter yet approximate worst score or best score bounds – to be used in Algorithm 1's line 32 – based on statistics about the proximity vectors.

Reason 2. The on-the-fly exploration of the social network in decreasing order of proximity may still place a significant execution overhead on query processors, even when relying on efficient max-priority queue structures. As our instance optimality result indicates, this is unavoidable in cases where exact computations are required. However, if approximate results are accepted, we can speed-up this step by techniques that yield approximate proximity values – replacing line 9 of Algorithm 1 – and visit the network accordingly.

Approximate score bounds and proximity scores may obviously lead to approximate final results. The techniques we present here provide a good trade-off between accuracy drop on one hand, and speed-up and the amount of necessary storage, on the other hand. We detail next our approaches for the two outlined directions.

4.1 Tighter score bounds estimations

In Algorithm 2, the MAXSCORE, MAXSCOREUNSEEN and MINSCORE bounds have all used the safest possible values for the proximities of yet unseen users: either the top (maximum) value of the max-priority queue ($top(H)$) for the first two bounds, or its minimal possible value (zero) for the third one. In practice, however, any of these extreme configurations is rarely met, and values in proximity vectors fall rapidly in many real-world networks.

Hence one possible direction for reducing the number of visited users is to pre-compute and materialize a high-level description of users' proximity vectors. This would allow us to use a tighter estimation for the remaining (unseen) users, instead of uniformly associating them the extreme score ($top(H)$ or 0).

Estimating bounds by mean and variance. We first consider as a proximity vector description one that is very concise yet generally-applicable and effective, keeping for a given seeker two parameters: the *mean* value of the proximities in the vector and the *variance* of these values. We adopt here the simplifying assumption that the values in any seeker's proximity vector are independent, essentially interpreting proximity vectors as random ones.

At any step in the algorithm's run, using mean and variance, for the remaining (unvisited) $unseen\_users(i, t)$ for a given item i and tag $t \in Q$, we can derive (a) lower bounds for the average of their proximity values, for MINSCORE estimates, or (b) upper bounds for the average of proximity values, for MAXSCORE estimates. The guarantees of these bounds can be controlled in a probabilistic sense by a precision parameter $\delta \in (0, 1]$, by which lower values lead to higher precision and 1 leads to absence of guarantees.

More precisely, let p be the current position in the proximity vector and let $\sigma^+_{p:}(s)$ be the vector containing the remaining (unseen) values of $\sigma^+(s)$. Knowing the overall mean and variance of the entire proximity vector $\sigma^+(s)$, and having the proximity values seen so far (denoted $\sigma^+_{0:p}(s)$), we can easily compute the mean and variance of the remaining proximity values (those in $\sigma^+_{p:}(s)$).

The mean and variance of $unseen\_users(i, t)$ randomly chosen proximity values from the remaining ones can be obtained – under an independence assumption – as follows:

$$Exp[\sigma^+_{p:}, unseen\_users(i, t)] \quad = \quad E[\sigma^+_{p:}],$$

$$Var[\sigma^+_{p:}, unseen\_users(i, t)] \quad = \quad \frac{Var[\sigma^+_{p:}]}{unseen\_users(i, t)}.$$

When the input query contains several tags, its size $|Q|$ needs to be taken into account in estimations. In order to avoid computational overhead, we uniformly chose a non-optimal per-tag probabilistic parameter δ' that ignores per-tag score distributions, as

$$\delta' = 1 - (1 - \delta)^{1/|Q|}. \tag{4.1}$$

$\text{ESTMAX}(p, \delta)$ represents, for each query tag, the upper bound of the expected value of the average of $unseen\_users(i, t)$ values drawn from $\sigma^+_{p:}(s)$, which holds with probability at least $1 - \delta'$. Similarly, $\text{ESTMIN}(p, \delta)$ represents the lower bound of the expected value of the average of $unseen\_users(i, t)$ values drawn from $\sigma^+_{p:}(s)$, which holds with probability at least $1 - \delta'$. For estimating MINSCORE when $i \notin CIL(t)$, having no information about the difference between $tf(i, t)$ and $partial\_tf(t, i)$ (the users who tagged item i with t so far) means that we cannot assume that other users may have tagged i, so we keep this estimation as in the initial (exact) algorithm. By Chebyshev's inequality, these bounds are

$$\text{ESTMAX}(p, \delta) \quad = \quad E[\sigma^+_{p:}(s)] + \sqrt{\frac{Var[\sigma^+_{p:}(s)]}{unseen\_users(i, t) \times \delta'}}$$

$$\text{ESTMIN}(p, \delta) \quad = \quad E[\sigma^+_{p:}(s)] - \sqrt{\frac{Var[\sigma^+_{p:}(s)]}{unseen\_users(i, t) \times \delta'}}$$

Estimating bounds using histograms. The advantage of the approach described previously is twofold: low memory requirements and estimated bounds that are applicable for any proximity distribution. However, it may still offer bounds that are too loose in practice, and hence not achieve the full efficiency potential of approximate score bounds.. To address this issue, we can imagine, as a compromise between keeping only these two statistics and keeping the entire pre-computed proximity vector, an approach in which we describe the distribution at a finer granularity, based on *histograms*.

More precisely, for seeker s, we denote this histogram as $h(\sigma^+(s))$. It consists of b buckets, and each bucket b_i, for $i \in \{1, \ldots, b\}$, has n_i items in the interval $(low_i, high_i]$ (the 0 values are assigned to

bucket b). Then, the probability that there exists a proximity value x greater than low_i, knowing the histogram $h(\sigma^+(s))$, is

$$Pr[x > low_i \mid h(\sigma^+(s))] = \sum_{j=1}^{i} n_j/n.$$

At any step p in the run of the algorithm, we maintain a partial histogram denoted as $h(\sigma^+_{p:}(s))$, obtained by removing from $h(\sigma^+(s))$ the p already encountered proximity values.

Similar to the previous approach, we can drill down the overall δ parameter to a δ' one for each query tag. Then, EstMax(p, δ) can be given by the minimal value in the partial histogram, such that the resulting estimation of MaxScore(i, t) holds with at least probability $1 - \delta'$. Conversely, EstMin(p, δ) is given by the maximal value in the partial histogram, such that the resulting estimation of MinScore(i, t) holds with at least probability $1 - \delta'$.

In manner similar to Eq.(4.1), we need to account for the fact that some $unseen\_users(i, t)$ such estimated values lead to an overall approximate estimation, for both EstMin and EstMax. So each of these values is uniformly estimated using a stronger probabilistic parameter $\delta''(i, t)$, depending on $unseen\_users(i, t)$, as

$$\delta''(i, t) = 1 - (1 - \delta'')^{1/unseen\_users(i,t)},$$

and hence we can estimate EstMax(p, δ) and EstMin(p, δ).

The space needed for keeping such histograms is linear in the number of users and buckets. For example, by setting the latter using the square-root choice, the memory needed is $O(n^{\frac{3}{2}})$. Also, as a consequence of the on-the-fly computation of proximity values, we can easily update the histogram of the seeker by merging the partial, "fresh" histogram obtained in the current run (until termination) with the remaining values from the pre-computed histogram.

4.2 Computing approximate user proximities

We present in this section our approach for the second approximate direction, by computing proximity values in approximate, efficient manner. For this, we adapt the landmark-based approach of Potamias et al. [17], which studies fast and accurate estimations of shortest paths in large graphs.

The principle of the approach is the following. Given a set $\mathcal{L}$ of nodes, called *landmarks*, we compute their proximity vectors (i.e., proximity w.r.t. all other nodes of the network). The number of landmarks can be considered to be a small constant, not depending on the network size (10 in our experiments). Then, for a given seeker s, by knowing her proximity values $\sigma^+(s, l_i)$ to each of the landmarks, we can use the triangle inequality to compute upper and lower bounds on the distance between s and any other node v, $\sigma^+(s, v)$. For instance, when path multiplication is used to compute proximity, these bounds can be obtained as follows:

$$\min(\frac{\sigma^+(s, l_i)}{\sigma^+(l_i, v)}, \frac{\sigma^+(l_i, v)}{\sigma(s, l_i)}) \geq \sigma^+(s, v) \geq \sigma^+(s, l_i) \times \sigma^+(l_i, v).$$

(Similar bounds can be obtained for any proximity computation function verifying Property 1.) The tightest proximity interval for v is then given by the minimal upper bound value over all landmarks and the maximal lower bound value over all landmarks.

Algorithm 3: GetMaxLandmarks$(s, \mathcal{L}, \{h_1, \ldots, h_{|\mathcal{L}|}\})$

Require: *seeker s, landmarks $\mathcal{L}$, vectors head pointers* $\{h_1, \ldots, h_{|\mathcal{L}|}\}$
1: $i \leftarrow argmax\{\sigma^+(l_i, h_i) \mid l_i \in \mathcal{L}, 1 \leq i \leq |\mathcal{L}|\}$
2: $u = h_i$
3: advance pointer h_i in l_i's list
4: $\sigma^+(s, u) = \max\{\sigma^+(s, l_j) \times \sigma^+(l_j, u) \mid l_j \in \mathcal{L}, 1 \leq j \leq |\mathcal{L}|\}$
5: $maxUB = \max\{\sigma^+(s, l_k) \times \sigma^+(l_k, h_k) \mid l_k \in \mathcal{L}, 1 \leq k \leq |\mathcal{L}|\}$
6: **return** u

We detail the computation of the next closest user in Algorithm 3, which replaces line 9 of Algorithm 1. We keep the $|\mathcal{L}|$ vectors ordered descending by the proximity scores. Let h_i denote the current (unvisited) user at the head of l_i's list. Then, at each step, we consider the unvisited user u having the highest proximity value among all the landmarks' proximity vectors. We advance the pointer in the respective vector, and then random-access u's score in each of the other landmarks' proximity vectors, obtaining $\sigma^+(s, u)$ as the maximal proximity upper bound. The proximity upper bound for yet unvisited users, $maxUB$ (to be used in the score estimations of line 32 in Algorithm 1, instead of $top(H)$) is obtained by computing the maximal distance from the seeker to any of the unvisited users; by monotonicity, it suffices to only look at the values at the heads of the $|\mathcal{L}|$ vectors. This operation is equivalent to one loop of the Threshold Algorithm (TA) [12], using the function `max` as aggregation.

Table 1: Comparison between algorithms, $\alpha = 0$.

Network	CMerge		SNS		SNS/H		SNS/L	
	users	CIL	users	CIL	users	CIL	users	CIL
tag	89k	0	88k	21	63k	8	29k	7
item	264k	0	185k	59	102k	24	212k	63
item-tag	240k	0	155k	39	104k	19	160k	47

5. EXPERIMENTAL EVALUATION

Dataset. In Twitter, a highly popular microblogging and social tagging application, one broadcasts to followers short messages (*tweets*). One can also re-broadcast incoming tweets from followees (*re-tweets*); when doing so, the re-tweet may be tagged with certain tags or short descriptions (*hashtags*). So we can see a tweet and its re-tweeted instances as representing one data item, which may be tagged with various tags by the users broadcasting it. This data can thus be modeled naturally as triples $(user, item, tag)$. Our dataset comes from a collection of tweets obtained via the public Twitter Streaming API. From the initial stream of tweets, we filtered out those that were never re-tweeted. Then, from the resulting set of triples, we filtered out those corresponding to (i) items not tagged by at least 2 distinct users, or (ii) users who did not tag at least 2 distinct items.

From this collection of triples, we generated three user similarity networks, in which the proximity between two users is given the Dice coefficient of either (i) the sets of tags used by those users (a *tag similarity* network), (ii) the sets of items tagged by those users (an *item similarity* network), or (iii) the sets of item-tag pairs of those users (an *item-tag similarity network*). The properties of this dataset and its similarity networks are as follows (as expected, all networks have long tail degree distributions and low diameter):

- 570,387 users; 1,570,866 items; 305,361 tags; 8,753,706 (u, i, t) triples.
- 10.10 distinct items per user; 1.39 distinct tags per item; 9.45 distinct tags per user.
- 324.1average degree (tag); 52.2 average degree (item); 52.6 average degree (item-tag).
- 0.41 average proximity (tag); 0.18 average proximity (item); 0.19 average proximity (item-tag).

Setup. We generated 20 top-10 queries, formed by 2 or 3 semantically coherent tags, from those having a medium frequency. For each network, 10 users were also randomly chosen in the seeker's role.

In the score model, the aggregation function g was `sum`. For the ranking function (the h-function), we used either the standard tf-idf: $score(i \mid u, t) = fr(i \mid u, t) \times idf(t)$, or BM15 as in [19]:

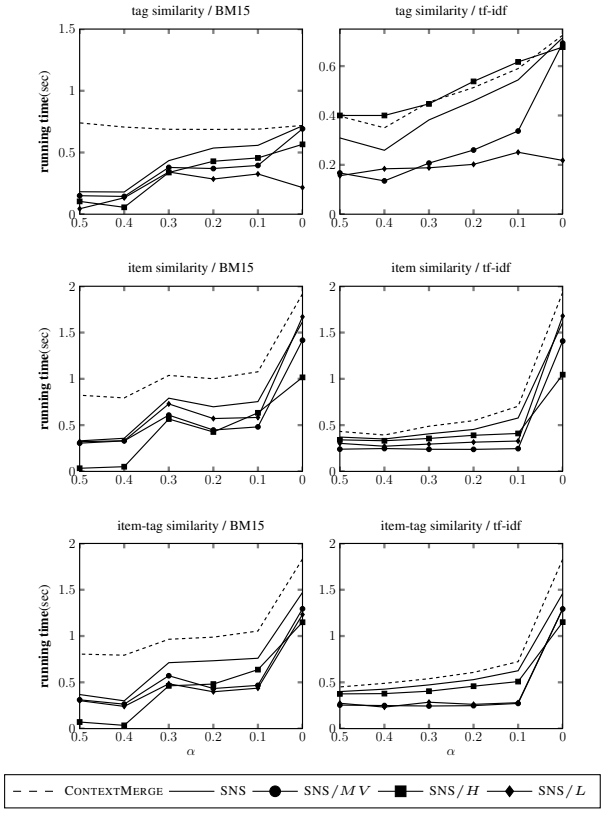

Figure 2: Efficiency results.

$score(i \mid u,t) = idf(t) \times (k1+1)fr(i \mid u,t)/(k1+fr(i \mid u,t))$, where inverse document frequency idf is defined in a standard way.

We ran Java implementations of our algorithms on a 2.8GHz Intel Core i7 under Ubuntu10.04, with 8GB of RAM, using PostgreSQL9.

As our focus is on optimizing social top-k retrieval, we report here on our results for the cases when the social branch is at least as important as the textual one, i.e., for $\alpha \in \{0, \ldots, 0.5\}$. As [19], multiplication over the paths was chosen as the proximity aggregation function, as the best suited candidate for implicit similarity.

For the testing environment described previously, we report on *efficiency* and *scalability* for both exact and approximate algorithms, and on *precision* for the latter. For efficiency (running time), we *ignore differences in* SNS*'s favor that are hard to account for*, namely we do not distinguish between the user accesses by ContextMerge (which in a real setting, would be to disk) and the ones by SNS (which could even be to main memory). For that, we plugged our on-the-fly computation of proximity in the run of ContextMerge. *Recall that an immediate consequence of Property 2 is that* SNS *(exact) and* ContextMerge *give the same results.*

The relevance of personalized query results is a topic that has been extensively treated ([20, 11, 16]). It is not our focus here, and we view result relevance as a consequence of the scoring functions g and h. The relevance of social search results was also extensively evaluated in [19], over Delicious data, in a similar setting.

Efficiency. Fig. 2 gives the running time comparison for tf-idf and BM15 scores. First, note that, in general, SNS in its exact version drastically improves efficiency – especially for low α values – when compared to the state-of-the-art algorithm. Note that our branch choice heuristic (in both the exact and approximate variants) brings

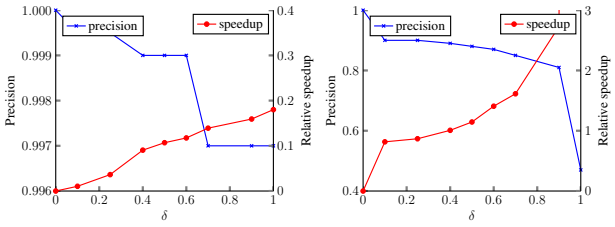

Figure 3: Precision rates vs relative speedup, $\alpha = 0$ – left: SNS$/MV$, right: SNS$/H$

significant improvements (e.g., consider the difference between the savings for $\alpha = 0$ and $\alpha = 0.1$, in the tag similarity network).[4]

Note that instance optimality is indeed not a synonym of efficiency in some cases (for example, in tag similarity networks). In these cases, our approximate approaches lead to further improvements, supporting the intuition that even limited statistics (like mean and variance) can render the termination conditions more tight.

The running times of SNS$/MV$ and SNS$/H$ were obtained for the probabilistic threshold $\delta = 0.9$. While this is a rather weak guarantee, we found that it still yields a good precision/efficiency tradeoff. (For a better understanding of this tradeoff, we consider later on the impact of δ on precision.) For $\alpha > 0$, visiting the per-term inverted lists in parallel with the proximity vector helps in deriving tighter bounds for unseen items, leading to faster termination of the approximate approaches. For the landmarks-based variant, consistent with [17]'s results, where choosing landmarks based on centrality is shown to yield precise estimations, we took as landmarks the 10 nodes having the highest centrality values.

To better understand how the instance optimality of SNS$_{\alpha=0}$ reflects in the performance results, Table 1 reports on the number of visited users (by thousands) by ContextMerge and SNS$_{\alpha=0}$ (columns *users*). Note that SNS$_{\alpha=0}$ achieves good savings (in terms of visited users), while relying on only few sequential accesses in the inverted lists (column *CIL*). SNS$/H$ can further decrease the number of visited users in item and item-tag similarity networks, and SNS$/L$ performs best in tag similarity networks.

Precision versus speedup. We studied the impact of the probabilistic parameter δ on precision and speedup, in the approximate algorithms. We define precision as the ratio between the size of the exact result and the number of items returned by both the approximate approach and SNS, i.e., $precision = |T_{\text{SNS}/app} \cap T_{\text{SNS}}|/|T_{\text{SNS}}|$. $T_{\text{SNS}/app}$ is the set of items returned as top-k by an approximate approach (SNS$/MV$, SNS$/H$ or SNS$/L$), and T_{SNS} is the set of items returned by the exact algorithm.

The relative speedup is defined as $speedup = time(\text{SNS}, D)/time(\text{SNS}/app, D) - 1$. We present in Figure 3 the results for the two approximate approaches based on statistics. For SNS$/MV$, note that δ has a limited influence on precision (with a minimum of 0.997 for $\delta = 1$), while ensuring reasonable speedup. The speedup potential is greater when using histograms, with reasonable precision levels (e.g., precision of around 0.805 when $\delta = 0.9$, for a speedup of around 2.5). For values of $\delta > 0.9$, we note however a rapid drop in precision. The fact that SNS$/MV$ yields better precision than SNS$/H$ may seem counterintuitive, since histograms give a more detailed description of proximity vectors. This difference is due to looser bounds for MV, as they directly influence the termination

[4]Note that we cannot compare with [1]'s approach, as it only extends classic top-k retrieval by interpreting user proximity as a binary function (0-1 proximity), by which only users who are directly connected to the seeker can influence the top-k result.

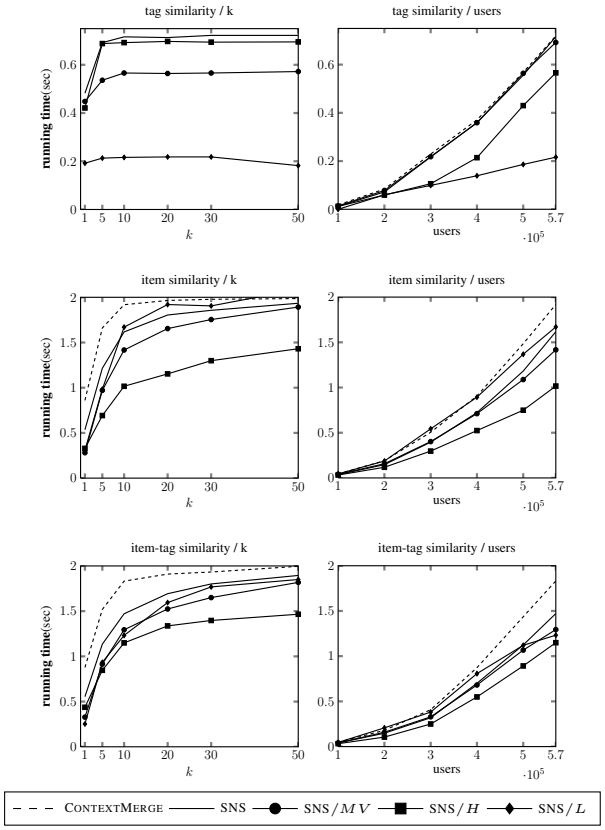

Figure 4: Scalability of the algorithms.

conditions, result in a longer run and better chances of returning a more precise results. For the landmark-based variant, SNS$/L$, we mention here that choosing 10 landmarks yields a precision above 0.65 in both datasets, when $\alpha = 0$, and above 0.95 for $\alpha > 0$. (We also observed that with more landmarks there is no precision improvement, but execution becomes slower.)

Scalability. We also evaluated how our approaches scale with two key parameters, namely the result size k and the network size (number of users), in the exclusively social case. As shown in the top row of Figure 4, the exact approach maintains relatively constant gains. Approximate approaches, especially SNS$/H$ and SNS$/L$, exhibit higher gains and their cost increases relatively slowly. The "plateau" in running times for high k values is mainly due to the fact that – as the number of retrieved items increases – the need for accessing more items in the textual lists and more users increases. However, beyond a certain point (here, around $k = 20$), the termination conditions become easier to satisfy, and hence the increase in cost is less sharp than for low k values. We observed a different behavior when varying the network size (bottom row of Figure 4). First, we can note a linear cost increase. The gains of our algorithms w.r.t. the state-of-the-art one increase with the network size, both for the exact and approximate versions. As with varying k, the approximate approaches exhibit a slower cost increase in most cases.

Wrapping up. It can be seen that, in most cases, instance optimality does indeed translate into significant performance gains. These are important when top-k results need to be exact, as in cases when pre-computed information (such as average values, histograms or shortest paths) is not available. On the other hand, when exact results are not needed, keeping partial aggregates such as histograms seems to work best in sparser networks, while approaches based on precomputed shortest paths (landmarks) seem to have a bigger impact in denser networks. Moreover, when the result size k is large and so the impact of "missed" items may be more limited, relying on histograms and landmarks seem the best options for efficiency.

6. FUTURE DIRECTIONS

We see many directions for future work. Optimizing the branch choice heuristic is one promising direction. Another one is experimenting with probabilistic bounds and statistics tailored to certain assumptions (e.g, power-law distributions) or richer descriptions for proximity vectors. Also, we intend to adapt our approach to networks with positive and negative links (e.g., for trust and distrust).

Acknowledgments. This work was partially supported by the EU project ARCOMEM FP7-ICT-270239.

7. REFERENCES

[1] S. Amer-Yahia, M. Benedikt, L. Lakshmanan, and J. Stoyanovich. Efficient network aware search in collaborative tagging sites. In *VLDB*, 2008.

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

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

[4] B. Bahmani and A. Goel. Paritioned multi-indexing: bringing order to social search. In *WWW*, 2012.

[5] S. Bao, G. Xue, X. Wu, Y. Yu, B. Fei, and Z. Su. Optimizing web search using social annotations. In *WWW*, 2007.

[6] B. Bi, S. Lee, B. Kao, and R. Cheng. An effective and efficient method for searching resources in social tagging systems. In *ICDE*, 2011.

[7] X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. In *PVLDB*, 2010.

[8] D. Carmel, N. Zwerdling, I. Guy, S. Ofek-Koifman, N. Har'el, I. Ronen, E. Uziel, S. Yogev, and S. Chernov. Personalized social search based on the user's social network. In *CIKM*, 2009.

[9] S. Cohen, B. Kimelfeld, G. Koutrica, and J. Vondrak. On the principles of egocentric person search in social networks. In *VLDS*, 2011.

[10] E. Dijkstra. Note on two problems in connexion with graphs. *Num. Mathematik*, 1959.

[11] Z. Dou, R. Song, and J. Wen. A large-scale evaluation and analysis of personalized search strategies. In *WWW*, 2007.

[12] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[13] P. Heymann, G. Koutrika, and H. Garcia-Molina. Can social bookmarking improve web search? In *WSDM*, 2008.

[14] A. Hotho, R. Jäschke, C. Schmitz, and G. Stumme. Information retrieval in folksonomies: Search and ranking. In *ESWC*, 2006.

[15] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 1953.

[16] I. Konstas, V. Stathopoulos, and J. Jose. On social networks and collaborative recommendation. In *SIGIR*, 2009.

[17] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis. Fast shortest path distance estimation in large networks. In *CIKM*, 2009.

[18] M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. In *ISWC*, 2003.

[19] R. Schenkel, T. Crecelius, M. Kacimi, S. Michel, T. Neumann, J. Parreira, and G. Weikum. Efficient top-k querying over social-tagging networks. In *SIGIR*, 2008.

[20] J. Wang, M. Clements, J. Yang, A. P. de Vries, and M. J. T. Reinders. Personalization of tagging systems. *Inf. Process. Manage.*, 2010.

[21] S. Xu, S. Bao, B. Fei, Z. Su, and Y. Yu. Exploring folksonomy for personalized search. In *SIGIR*, 2008.

[22] P. Yin, W.-C. Lee, and K. C. Lee. On top-k social web search. In *CIKM*, 2010.