Efficient Top-k Querying over Social-Tagging Networks

Ralf Schenkel‡ Tom Crecelius‡ Mouna Kacimi‡ Sebastian Michel‡ Thomas Neumann‡
Josiane Xavier Parreira‡ Gerhard Weikum†

†Max-Planck Institute for Informatics
Saarbrücken, Germany

‡École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland

{schenkel, tcrecel, mkacimi, neumann, jparreir, weikum}@mpi-inf.mpg.de
sebastian.michel@epfl.ch

ABSTRACT

Online communities have become popular for publishing and searching content, as well as for finding and connecting to other users. User-generated content includes, for example, personal blogs, bookmarks, and digital photos. These items can be annotated and rated by different users, and these social tags and derived user-specific scores can be leveraged for searching relevant content and discovering subjectively interesting items. Moreover, the relationships among users can also be taken into consideration for ranking search results, the intuition being that you trust the recommendations of your close friends more than those of your casual acquaintances.

Queries for tag or keyword combinations that compute and rank the top-k results thus face a large variety of options that complicate the query processing and pose efficiency challenges. This paper addresses these issues by developing an incremental top-k algorithm with two-dimensional expansions: social expansion considers the strength of relations among users, and semantic expansion considers the relatedness of different tags. It presents a new algorithm, based on principles of threshold algorithms, by folding friends and related tags into the search space in an incremental on-demand manner. The excellent performance of the method is demonstrated by an experimental evaluation on three real-world datasets, crawled from deli.cio.us, Flickr, and LibraryThing.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.3.3 [**Information Search and Retrieval**]: Search Process

General Terms

Performance, Experimentation

Keywords

social networks, scoring and ranking, top-k query processing

1. INTRODUCTION

The advent of Web 2.0 has changed the way users interact with the Internet. Publishing content has never been easier, and new online services offer not only the possibility to store personal content, but also to share it with other people and to explore other users' contents. Examples of social-community platforms are del.icio.us, Flickr, LibraryThing, LinkedIn, MySpace, Facebook, and YouTube.

While differing in the type of content that they focus on (e.g., blog entries, photos, videos, books, bookmarks), most community platforms follow a common pattern. *Users* must register in order to join the community. Then they produce information, ideally by publishing their own *documents*[1] and by adding *tags* or ratings or comments to other content already available in the community. The platforms also offer a way to maintain a list of *friends* and means to keep friends informed about your latest content items. The size of your friend network is often considered as an indication of high reputation in the network. Many users initially populate their lists of friends with people they already know from the offline world or other online communities, but over time they typically identify previously unknown users that they share common interests with and also add those users to their friends lists.

A key feature of social communities is the widely used opportunity to attach manually generated annotations, so-called *tags*, to content items. Tags can provide precise descriptions of content items, flavored with the respective personal interest of the user who generates the tag.

Social tagging offers an opportunity to exploit the "wisdom of the crowds" by identifying valuable content that is recommended by friends - either directly or indirectly (i.e., via friends of friends) and explicitly (e.g., ratings) or implicitly (e.g., by intensive tagging). To this end, the relationships among users should be taken into account for ranking, the intuition being that you trust the recommendations of your close friends more than those of your casual acquaintances. This situation resembles the paradigm of collaborative recommendation [19, 25], which applies data mining on customer-product and similar usage data to predict items that users are likely interested in. However, the fast growth of communities and the very high rate of content production and tagging efforts calls for highly *efficient* and *scalable* methods. Existing algorithms for fast Web search do not consider user relationships and the assets from social tagging, and prior methods for collaborative recommendation do not provide the throughput scalability that one needs for running millions of daily ad-hoc queries in social communities such as Flickr - with more than 2 billion content items, many million users, and high dynamics.

This paper presents a solution to the above problem by developing an efficient top-k algorithm for social search and ranking. The method is based on principles of top-k threshold algorithms over inverted lists of different types, with various novel techniques for the specific setting of large online communities. For leveraging

[1]For the remainder of this paper, we will use the familiar IR-jargon term "document", while actually referring to a more general notion of content items.

SIGIR'08, July 20–24, 2008, Singapore.

the "social wisdom", the algorithm employs a new form of two-dimensional expansions: social expansion considers the strength of relations among users, and semantic expansion considers the relatedness of different tags. For efficiency, these expansions are performed incrementally on demand, by dynamically folding friends and related tags into the search space. The excellent performance of the method is demonstrated by an experimental evaluation on three real-world datasets, crawled from deli.cio.us, Flickr, and LibraryThing.

The rest of the paper is organized as follows. Section 2 reviews related work and puts it into context. Section 3 introduces a graph model to represent the entities in a social community and their relationships. Section 4 presents ways of quantifying the strengths of different relationships, which in turn feeds into a model for computing scores of user-specific search results and recommendations. Section 5 explains the algorithm for efficiently processing queries in social networks. Section 6 presents experiments with three real-world datasets to study retrieval/recommendation effectiveness and, most notably, demonstrate the efficiency gains by the new query processing algorithm.

2. RELATED WORK

Social networks of the Web 2.0 style have received major attention in the recent literature, with focus on applying data mining methods on social relations and, most prominently, relations among tags. [16] provides an empirical study of the tagging behavior and tag usage in online communities. [21, 31] discuss methods for generating taxonomy-like relations among tags, so-called "folksonomies", based on statistical measures. Similar approaches have been applied to query-and-click logs, e.g., in [5], but none of this work considers social relations between different users. Identifying important and emergent tags and visualizing them in so-called "tag clouds" (and corresponding time series) has been extensively explored; recent work along these lines includes [14, 17, 32]. The dynamics of social relations among users (e.g., the rate of making friends) has been studied, for example, in [2, 24, 33].

As for the exploitation of social tags for information retrieval, [3] discusses the challenges of searching and ranking in social communities. Various forms of community-aware ranking methods have been developed, mostly inspired by the well-known PageRank method [10] for web link analysis. [22] proposes *FolkRank* for identifying important users, data items, and tags. [36] compares different methods for identifying authoritative users with high expertise. [6] introduces *SocialPageRank*, to measure page authority based on its annotations, and *SocialSimRank* for the similarity of tags. [35] further extends this work by augmenting language models with tag similarities. [13] shows that explicit user tagging can help to improve precision of queries for Intranet search. The very recent work of [20] provides an empirical analysis of how social bookmarking can influence Web search, with both positive and negative insights. None of this prior work considers the impact of user-friendship strengths on the scoring of search results, and the problem of efficient query processing in the presence of such "social wisdom".

Aspects of user communities have also been considered for peer-to-peer search, most notably, for establishing "social ties" between peers and routing queries based on corresponding similarity measures (e.g., similarities of queries issued by different peers). [8] has studied "social" query routing strategies based on explicit friendship relationships and behavioral affinity. [27] has developed an architecture and methods for "social" overlay networks that connect "taste buddies" with each other. [26] has proposed a community-enhanced Web search engine that takes into account prior clicks by community members. [11] has proposed the notion of Peer-Sensitive ObjectRank, where peers receive resources from their friends and rank them using peer-specific trust values.

There is ample literature on collaborative filtering for recommender systems (e.g., [1, 18, 29, 30]), for example, to predict movies or e-commerce items that customers are likely to buy or to identify news that news-feed subscribers are likely interested in. In a nutshell, these methods aim to learn user preferences from the collective behavior - like purchases or tagging - of an entire community. Typically, statistical analysis and machine learning techniques are used offline for precomputations, and the actual runtime recommendations have limited flexibility and cannot easily cope with high dynamics and ad-hoc interests of individual users (as expressed by an ad-hoc query). One of the notable exceptions is the recent work by [12] which addresses scalability issues when the number of users and items in a recommender system grows to many millions and both undergo fast changes. However, in contrast to our "social search" theme, this prior work considers only the space of user-item pairs and there is no notion of user-specific tags or annotations on items. Thus, our setting requires search over a three-dimensional user-tag-item space, as opposed to the two-dimensional user-item space of the previous work on collaborative filtering.

3. SOCIAL NETWORK MODEL

The entities that occur in social networks can be cast into a common graph model, representing the different elements of a social network and their mutual relationships. Figure 1 illustrates the graph, which forms the basis of our scoring model and query processing algorithm.

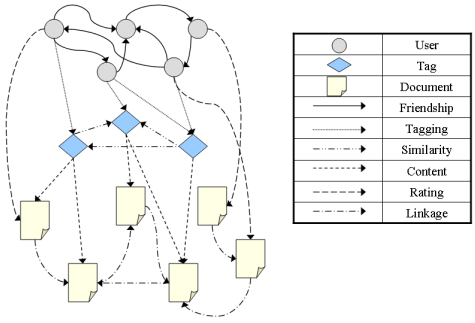

Figure 1: Social Network Model

A node in the graph can be of one of the following types: it can either represent a **user**, a **tag**, or a **document** (data item). Additionally, social networks exhibit various relationships, both among the nodes of the same type and between nodes of different types; these are represented by edges in the graph and described below. Edges can be weighted in different ways, according to the applications built on top of the model.

3.1 Relations Among Nodes of the Same Type

The following relations exist between nodes of the same type:

Friendship(User1, User2, FriendshipStrength): Social networks allow users to maintain an explicit list of friends. The trust of one user in another user (as far as potential recommendations are concerned) is reflected in the FriendshipStrength. (The quantitative measures to this end are discussed in Section 4.) Additional means of establishing implicit friendships include, for example, a user subscribing to another user's content (e.g., adding the other user's books to your set of InterestingLibraries in LibraryThing) or subscribing to the same user group as another user expressing high overlap in thematic interests. Regardless of how direct friends

are defined, we may consider the transitive closure of the Friendship relation or a bounded set of transitive connections (up to some distance).

TagSimilarity(Tag1, Tag2, TagSim): As tags are freely selected annotations and there is often a natural diversity in the tag wordings used by different users in a social network, different tag words may express (near-)synonyms (e.g., "feline" and "cat", "Web_2.0" and "Social_Web", etc.) or other kinds of semantically related concepts (e.g., hyponyms such as "dogs" and "German_shepherd", "search_engine" and "Google", etc.). The tag-usage statistics in the social network can be harnessed to derive TagSim similarities between tags (see Section 4 for quantitative measures).

Linkage(Document1, Document2, Weight): In some applications, documents (data items) also exhibit relations among themselves. In the case of web pages, this linkage is obvious and given by the hyperlink graph, with weights often chosen proportionally to the outdegree of the pages. For other types of documents, different notions of links and weights need to be defined; conceivable options could include, for example, the geographic proximity of different photos when GPS information is available.

3.2 Relations Among Nodes of Different Types

The following relations exist between different nodes types:

DocContent(Document, Tag, ContentScore): By annotating a document with a tag, users associate the two entities so that the tag should be viewed as an indicator of the document's content. We consider a ContentScore associated with a document-tag pair to reflect how well that tag describes the document.

Tagging(User, Tag, TagScore): Each tag is associated with each user who used it on at least one document; this captures the topics that the user is interested in. The TagScore reflects how intensively a tag has been used by one user.

Rating(User, Document, RatingScore): in many social communities, users can explicitly rate documents, which is captured by a rating score. Another naive instantiation of Rating is authorship of a content item, which (e.g., in the case of bookmarks) can be seen as an endorsement for the document.

Additional aspects can be considered as attributes associated with edges. Our model aims to capture all relationships that occur in social networks, but some of the above relationships may be undefined for specific networks. In del.icio.us, for instance, user interactions are mainly through bookmarking and tagging, whereas in Flickr, the vast majority of users has authored content (their photos).

For the purpose of search result ranking and top-k query processing, we will mainly focus on the Friendship and Tag Similarity scores, as discussed next.

4. SOCIAL SCORING MODEL

In line with the free-text tagging of the social communities, we consider a *query* $Q(u, q_1 \dots q_n)$, issued by a query initiator u, as a set of tags (keywords) $q_1 \dots q_n$, possibly with weights for each tag (which will be disregarded for the sake of presentation simplicity). Result documents should contain at least one of the query terms and be ranked according to a query-specific document *score*. In contrast to standard IR query models, our document scores also contain a social component: the content-based score of a document is additionally *user-specific*, i.e., it depends on the social context of the query initiator. This resembles the personalization approaches offered by some search engines, but social networks have the additional asset of knowing friendship relations and the friends' tagging behavior and are thus the most natural habitat to further explore and improve this idea.

Our social scoring model extends the traditional IR scoring models (tf-idf-based, probabilistic IR, language models) to search in social networks, with the following ingredients: (1) a measure for the importance of users, relative to the querying user, (2) a context-specific tag frequency relative to the querying user that reflects the relative importance of users which used a tag, and, optionally, (3) the expansion of query tags with related ("semantically similar") tags. In the following, we denote by U the set of users, by D the set of documents, and by T the set of tags.

Friendship Similarity. The importance of a user, relative to the querying user, is quantified by the *friendship similarity function* $F_u(u')$. We define $F_u(u) = 0$ (as a user is not interested in getting recommendations to her "own" documents/bookmarks which she knows anyway), and we normalize the F values by setting $\sum_{u' \in U} F_u(u') = 1$ for all users $u \in U$. $F_u(u')$ may be viewed as the probability that a random document that was tagged by u' will be interesting to u. The friendship similarity may be a combination of syntactic measures (like overlap of tag usage), social measures (like distance in the friendship graph), and global measures (like a PageRank-style importance of users computed on the friendship graph). In our implementation, we first compute an overlap-based similarity $O(u, u')$ for directly connected users in the friendship graph (*direct* friends) as the Dice coefficient of the sets of tags u and u' used:

$$O(u, u') = \frac{2 \times |tagset(u \wedge u')|}{|tagset(u)| + |tagset(u')|}.$$

We then extend this to users u and u' that are indirectly connected by one or more friendship paths by aggregating similarities along each path and picking the path with highest similarity. To this end, similarities can be averaged, multiplied, or multiplied weighted by (linear or dampened) distance, etc. In the implementation, we use

$$P_u(u') = \max_{path\ u=u_0 \dots u_k=u'} \prod_{i=0}^{k-1} O(u_i, u_{i+1})$$

which favors users at small distances. Other types of direct friendship strength and aggregation over paths can be easily plugged into the model and our implementation.

As not all users are connected in the friendship graph, we additionally use a uniform background model to assign a small, constant similarity also to unconnected users; so the final definition of friendship similarity is:

$$F_u(u') = \alpha \cdot \frac{1}{|U|} + (1 - \alpha) \cdot P_u(u')$$

Social Frequency. To reflect the similarity of the users who tagged a document that may be of interest to the querying user, we introduce the notion of *social frequency*, which replaces the standard term frequency (tf) and considers friendship similarities. More formally, denoting by $tf_u(d, t)$ the number of times user u used tag t for document d, we define the *social frequency* $sf_u(d, t)$ for a tag t and a document d, relative to a user u, as

$$sf_u(d, t) = \sum_{u' \in U} F_u(u') \cdot tf_{u'}(d, t).$$

Note that $tf_u(d, t)$ is typically 1 (tagged once) or 0 (not tagged at all) in most of today's social-tagging platforms, but it is conceivable that quantitative ratings are factored into this measure or user feedback leads to non-binary sf values. By plugging the definition of $F_u(u')$ into the above formula, we obtain

$$sf_u(d, t) = \sum_{u' \in U} \left(\frac{\alpha}{|U|} \cdot tf_{u'}(d, t) + (1 - \alpha) \cdot P_u(u') \cdot tf_{u'}(d, t)\right)$$

showing us that the social frequency can be split into a *global* part $\sum_{u' \in U} \alpha t f_{u'}(d,t) = \alpha TF(d,t)$ that is independent of the querying user and corresponds to a weighted global term frequency $TF(d,t)$, and a user-specific frequency (the second sum) that depends on the friendship strengths of the querying user. We will make use of this decomposition for more efficient query processing in Section 5.

Social Score for Tags. To compute the score $s_u(d,t)$ of a document d with respect to a single tag t relative to the querying user u, we use a scoring function in the form of a simplified BM25 [28] score:

$$s_u(d,t) = \frac{(k_1+1) \cdot |U| \cdot sf_u(d,t)}{k_1 + |U| \cdot sf_u(d,t)} \cdot idf(t)$$

where k_1 is a tunable coefficient (just like in standard BM25) and $idf(t)$ is the inverse document frequency of tag t, instantiated as

$$idf(t) = \log \frac{|D| - df(t) + 0.5}{df(t) + 0.5}$$

with $df(t)$ denoting the number of documents that were tagged with t by at least one user. Unlike the original BM25 formula, our model has no notion of document lengths; the number of tags assigned to a document does not vary as much as the length of text documents.

Plugging the definitions of $sf_u(d,t)$ and $F_u(u')$ into the formula, we obtain $s_u(d,t) =$

$$\frac{(k_1+1)(\alpha TF(d,t) + (1-\alpha)\sum_{u' \in U} |U| P_u(u') t f_{u'}(d,t))}{k_1 + \alpha TF(d,t) + (1-\alpha)\sum_{u' \in U} |U| P_u(u') t f_{u'}(d,t)} \cdot idf(t)$$

As a special case, we can emulate socially agnostic global scoring, with just considering global tag frequencies $TF(d,t)$ and $idf(t)$, by setting $\alpha = 1$.

Tag Expansion. Even though related users are likely to have tagged related documents, they may have used different tags to describe them. It is therefore essential to allow for an expansion of query tags to "semantically" related tags. A simple way to account for this would be to statically expand the query with a fixed number of similar tags; however, experiments on text IR have shown that this can lead to topic drift and search results that are inferior to those of the unexpanded, original query [9]. Our scoring model therefore adopts the *careful expansion* approach proposed in [34] that considers, for the score of a document, only the best expansion of a query tag, not all of them. More formally, we introduce the *tag similarity* $tsim(t_1,t_2)$ for a pair of tags t_1 and t_2, $0 \leq tsim(t_1,t_2) \leq 1$. The final score $s^*_u(d,t)$ of a document d with respect to a tag t and relative to a querying user u, considering tag expansion, is then defined as

$$s^*_u(d,t) = \max_{t' \in T} tsim(t,t') \cdot s_u(d,t')$$

In our current implementation, the similarity between two tags is determined by the co-occurrence of the tags in the entire document collection by estimating conditional probabilities:

$$tsim(t,t') = P[t'|t] = \frac{df(t \wedge t')}{df(t)}$$

where $df(t \wedge t')$ is the number of documents that have been tagged by both tags (but possibly by different users). Other measures such as *SocialSimRank* from [6] could be easily incorporated as well.

Social Score for Queries. Finally, the score for an entire query with multiple tags $q_1 \ldots q_i$ is the sum of the per-tag scores:

$$s^*_u(d, q_1 \ldots q_n) = \sum_{q_1 \ldots q_n} s^*_u(d,q_i)$$

Note that this score assumes a non-conjunctive query evaluation. However, it can easily be extended to conjunctive evaluation by setting $s^*_u(d, q_1 \ldots q_n) = 0$ when at least one of the $s^*_u(d,q_i) = 0$.

5. QUERY PROCESSING

This section introduces the CONTEXTMERGE algorithm to efficiently evaluate the top-k matches for a query, using the social-context score introduced in the previous section. This algorithm generally falls into the well-established framework of threshold algorithms over impact-ordered inverted lists [15, 4] for efficient top-k query processing. However, as the social score depends on the user who submits a query, it is impossible to precompute per-tag scores for each document and each user. Standard algorithms that rely on scanning inverted lists cannot be applied here. Instead, CONTEXTMERGE makes use of information that is available in social tagging systems anyway, namely lists of documents tagged by a user and numbers of documents tagged with tags. It incrementally builds social frequencies by considering users that are related to the querying user in descending order of friendship similarity, computes upper and lower bounds for the social score from these frequencies, and stops the execution as soon as it can be guaranteed that the best k documents have been identified.

5.1 Preprocessing

CONTEXTMERGE makes use of four different kinds of preprocessed index lists that are built at indexing time and accessed mostly sequentially in descending order of scores at querying time. Additionally, random accesses to look up the value of an item in a list are possible, but more expensive than sequential accesses in terms of access cost.

- DOCS(t) contains, for a tag t, the documents d tagged by at least one user with tag t and the corresponding *global* tag frequencies $TF(d,t)$ (the total number of times all users together have tagged d with t), sorted by descending $TF(d,t)$.
- USERDOCS(u,t) contains, for a user u and a tag t, the (unsorted) set of documents d tagged by u with t and their user-specific tag frequency $tf_u(d,t)$ (the number of times u tagged d with t, often 1 for today's social-tagging platforms).
- FRIENDS(u) contains, for a user u, all related users u' and their similarity $P_u(u')$, sorted by descending $P_u(u')$.
- SIMTAGS(t) contains for a tag t all similar tags t' with their similarity $tsim(t,t')$, in descending order of $tsim(t,t') \cdot idf(t')$.

5.2 Algorithmic Framework

Figure 2 shows the general structure of our query processing framework. To compute the top-k results for a query $q_1 \ldots q_n$ submitted by a user u, CONTEXTMERGE sequentially scans, for all query tags, the DOCS lists and the USERDOCS lists of the friends of u in an interleaved way, maintains a list of candidate documents seen during the scans and a list of current top-k candidates, and terminates as soon as none of the candidates can move to the top-k. To improve efficiency, CONTEXTMERGE can additionally perform random accesses to the index lists to lookup the values for selected documents. Note that the algorithm can be further optimized if α is set to extreme values: For $\alpha = 0$, no DOCS lists need to be opened as the execution can be limited to the context of u; analogously, if $\alpha = 1$, there is no need to consider any lists of friends, so just the DOCS lists are read and CONTEXTMERGE behaves like a standard top-k algorithm. However, the interesting case is for α between 0 and 1.

Sequential Scans. To limit the number of disk accesses, the USERDOCS lists are opened only on demand, namely when the

```
procedure ContextMerge(user u, query q_1 ... q_n, α)
    for i ← 1 ... n do
        FRIENDS[i] ← FRIENDS(u)
        DOCS[i] ← DOCS(q_i)
    end for
    candidates ← ∅
    repeat
        for b ← 1 ... batchsize do
            l ← ChooseNextList( )
            if l = FRIENDS[i] then
                read USERDOCS(FRIENDS[i], q[i])
                go to next entry in FRIENDS[i]
            else if l = DOCS[i] then
                read DOCS[i]
            end if
        end for
        CheckRandomAccesses( )
        if CheckTermination( ) then
            break
        end if
    until termination
end procedure
```

Figure 2: ContextMerge without tag expansion

score that can be read from a USERDOCS list is greater than the score from any DOCS list. The algorithm additionally scans, for each query term, the list FRIENDS(u) of u's friends to determine an upper bound for the score that can be read from the USERDOCS list of the next friend for the query term. In each iteration of the main loop, ContextMerge does a batch of $batchsize$ list accesses, where in each access a number of entries is read from the list that can contribute the highest score. As we do not keep precomputed scores in the list, but just frequencies, we need to compute the score contributions from each list at run-time. To do this, the algorithm maintains the last value $high[i]$ read from each DOCS[i] list (which is set to the top value in the list upon initialization and updated when tuples are read from the list) and the last value $highF[i]$ read from the FRIENDS list in each query dimension (i.e., original query tag). The upper bound for the next score read from the DOCS[i] list can be computed by evaluating $s_u(d,t)$ with $TF(d,t) = high[i]$ and setting the user-specific part to 0. Analogously, the upper bound for the next USERDOCS list in dimension i (which could be read if the USERDOCS list for the next user in that dimension were opened) is computed as $\frac{(k_1+1)\cdot(1-\alpha)|U|highF[i]\cdot maxtf(q_i))}{k_1+|U|\cdot((1-\alpha)highF[i]\cdot maxtf(q_i))}\cdot idf(t)$, where $maxtf(q_i)$ is the maximal term frequency of any user and any document for tag q_i. The idf values needed to compute these scores are fetched once during the initialization of the execution.

The ChooseNextList method of the algorithm then greedily selects the list which has the highest expected score. If this is a FRIENDS list, the corresponding USERDOCS list is read completely and the FRIENDS list in that dimension is advanced by one. Note that in most of today's social-tagging applications, all documents in the USERDOCS list will have the same tf value of 1, so all of them are read in one shot. If the next list is a DOCS list, a configurable number of entries is read from it (as DOCS lists are usually much longer than USERDOCS lists).

Candidate Management and Termination Test. When scanning the index lists, ContextMerge collects candidates for the query result and maintains them in two disjoint priority queues, one for the *current top-k items* and another one for *all other candidates* that could still make it into the final top-k. For each candidate d_j, the algorithm maintains the following information:

- $TF(d_j, q_i)$, the value read for d_j from the DOCS lists for each query tag q_i
- $uf(d_j, q_i) = \sum_{USERDOCS(u')\ read\ for\ q_i} P_u(u')\cdot tf_{u'}(d_j, q_i)$, the weighted sum of values read for d_j from the USERDOCS lists for each q_i
- $E(d_j) \subseteq \{1 \ldots n\}$, the set of evaluated DOCS dimensions in which d_j has already been evaluated,
- $r(d_j, q_i)$, the number of times d_j has been seen in user lists for q_i,
- *worstscore*(d_j), a lower bound for the total score of d_j which is computed from the values seen so far for d_j,
- *bestscore*(d_j), an upper bound for the total score of d_j.

To compute the worstscore of a candidate, $s_u(d_j, q)$ is evaluated by setting $TF(d_j, q_i) = 0$ for $i \notin E(d_j)$ and using $uf(d_j, q_i)$ instead of the summation $\sum_{u' \in U} P_u(u') \cdot tf_u(d_j, q_i)$. Note that in conjunctive evaluation, the worstscore of a candidate remains 0 until the document has, for each query tag q_i, been seen in DOCS(i) (by sequential or random access) or in one of the USERDOCS lists read for q_i.

To compute d_j's bestscore, we assume $TF(d_j, q_i) = high[i]$ for $i \notin E(d_j)$ (the current upper bound of the corresponding DOCS list). Additionally, we need to estimate the contribution C from users that have not yet been seen, and use the sum $uf(d_j, q_i) + C$ in the social-context part of $s_u(d_j, q)$. As the algorithm considers users in descending order of similarity, we know that for any user u' that has not yet been considered for q_i, $P_u(u') \leq highF[i]$. Denoting by $mass_i = \sum_{USERDOCS(u')readfor q_i} P_u(u')$ the sum of the similarities of users already considered for q_i and by $maxtf$ the maximal tag frequency of any user for any document and tag in the collection (which is usually 1), we can estimate the remaining contribution as $C = (1 - mass_i)maxtf$, because the P_u values are normalized to a sum of 1. Moreover, if we additionally know the maximal number $maxu[i]$ of users who tagged any document with q_i, the maximal contribution from unseen users for d_j and q_i can be $C = (maxu[i] - r(d_j, q_i)) \cdot maxtf$ ($maxu[i]$ can be read during the initialization of the algorithm; the initial high value of the corresponding DOCS list is an upper bound for $maxu[i]$ which is tight if $maxtf = 1$).

The bestscore estimation can be made more precise if $TF(d_j, q_i)$ is known, because we can replace $maxu[i]$ in the formula by $TF(d_j, q_i)$ which often is much smaller. If $\alpha > 0$, i.e., if the algorithm scans the DOCS lists, we can replace $maxu[i]$ by $high[i]$ if $TF(d_j, q_i)$ is not yet known (if it was higher than $high[i]$, it would have already been read during the scan of the corresponding DOCS list).

In addition, the following information is derived at each step:

- the minimum worstscore $min\text{-}k$ of the current top-k docs, which serves as the stopping threshold, and
- the bestscore that any currently unseen document can get, which is computed by assuming that such an unseen document d_v (for *virtual* document) appears right at the front of the not yet seen parts of the lists. Its best score is then computed by setting $TF(d_v, t_i) = high[i]$, estimating the the contribution from not yet seen users like explained above, and evaluating the score in each dimension with these numbers.

The algorithm can safely terminate, yielding the correct top-k results, when the maximum bestscore of the candidate queue and the best score of any unseen document is not larger than $min\text{-}k$. Additionally, whenever a candidate in the queue has a bestscore that

is not higher than $min\text{-}k$, this candidate can be pruned from the queue (which helps to keep the memory footprint of the execution low as unneeded candidates can be removed early in the process). To further limit the CPU overhead of testing the candidates, this test is only performed after a full batch of scan steps, and only enabled after the bestscore of the unseen document d_v is not greater than $min\text{-}k$.

Random Accesses. In addition to sequential accesses (SA) to the index lists, CONTEXTMERGE can also perform random accesses (RA) to the index lists to look up missing scores of candidates. However, it is not feasible to check all USERDOCS lists of not yet seen users for a document, as this would require to explore the full range of (potentially many thousands of transitive) friends. Therefore, the only type of RA applied by CONTEXTMERGE is RA to DOCS lists to look up the global term frequency of a document for a tag. This serves two purposes: first, it can reduce the gap between the document's worstscore and bestscore because TF is then known exactly for one more tag; second, the estimation of the score contribution from the remaining friends gets more precise (as $TF(d,t)$ can be used in the estimation instead of $maxtf(t)$).

As RA are much more expensive than SA (in the order of 100 to 1,000 times for real systems), they have to be carefully selected and scheduled to avoid any unnecessary work. Our scheduling for RA follows the LAST heuristics from [7]: our algorithm performs only SA until the estimated cost to perform *all* RA to remaining candidates is at most as high as the cost for all SA done so far. We estimate the number of RA by summing up, for all candidates, the number of query dimensions (i.e., original query tags) that have not yet been evaluated.

5.3 Including Tag Expansion

Tag expansion adds another dimension that CONTEXTMERGE needs to combine with the user-expansion dimension. Conceptually, we add, for each similar tag t_{ij} of a query tag q_i, a new $DOCS[i][j]$ list (which corresponds to $DOCS(t_{ij})$ and a list $FRIENDS[i][j]$ of similar users. The score for these lists is computed like the score for lists without tag expansion, but additionally multiplied with $tsim(q_i, t_{ij})$. Candidate documents now maintain, for each query tag q_i, the information shown above not only for q_i itself but also for each similar tag t_{ij}. Following the max-semantics of our tag-expansion score, we can now estimate worstscores and bestscores of candidates as the maximum worstscore and bestscore for each similar tag, again weighted by $tsim(q_i, t_{ij})$.

However, it would be very inefficient to directly include the lists of all similar tags in the processing. Instead, CONTEXTMERGE limits the number of expansion tags per original query tag (e.g., by a limit of 10) and incrementally adds lists for similar tags to the processing on the fly. To do this, the algorithm maintains, for each query tag q_i, the list $SIMTAGS(q_i)$ of tags similar to q_i, and includes these lists in the list selection process (CHOOSENEXTLIST). To compute the score bounds of a list $SIMTAGS(q_i)$, CONTEXTMERGE first reads the entry $t_{ij}, tsim(q_i, t_{ij})$ from the top of the list without actually removing it from the list and looks up $idf(t_{ij})$. The score of the $SIMTAGS(q_i)$ list is then the maximum of the scores that the $DOCS$ and $FRIENDS$ lists for t had if they were opened for scanning. If CHOOSENEXTLIST chooses the $SIMTAGS(q_i)$ list, it adds $DOCS[i][j]$ and $FRIENDS[i][j]$ to the processing.

In this dynamic handling of tag expansions, the computation of worstscores for candidates must take into account that the actual score of a document for a query tag q_i is the maximum over all similar tags t_{ij} whose lists are open. In addition, the bestscore computation needs to consider that not all lists for expanded tags may have been opened already. For each query tag q_i where the tag expansion limit has not yet been reached, it therefore computes the maximal score that *any* document can get for the top tag t_{ij} in $SIMTAGS(q_i)$, and the bestscore of a document is then the maximum bestscore it can obtain in all open lists and the next expansion. Note that this bound is only correct because the entries in $SIMTAGS(q_i)$ are sorted by $tsim(q_i, t_{ij} \cdot idf(t_{ij}))$.

6. EXPERIMENTS

6.1 Data Collections and Benchmark Queries

We evaluate the effectiveness of our scoring model and the efficiency of the CONTEXTMERGE algorithm on three different datasets, crawled from social network websites of different kinds:

- **del.icio.us:** We have partly crawled the social bookmark site del.icio.us (http://del.icio.us/) with a total of 12,389 users, 175,754 bookmarks with 2,781,096 tags, and $152,306$ friendship connections.
- **Flickr:** We have partly crawled Flickr (http://flickr.com/) with a total of 52,347 users, 10,000,000 images with 29,111,183 tags, and 1,293,777 friendship connections. In addition to explicit friends defined by the user (which rarely happens in Flickr), we also considered two users as friends if one of them commented a picture uploaded by the other.
- **LibraryThing:** We have partly crawled the social book catalog LibraryThing (http://www.librarything.com) with a total of 9,986 users, 6,453,605 books with 14,295,693 tags, and 17,317 friendship edges. As users in LibraryThing typically use tags that consist of multiple terms, we extracted the terms from the tags and considered the set of terms used by a user for a book as if she tagged the book with the terms. The friendship here is again defined in a broader way, and consists of explicit friends and users marked as having interesting libraries.

Finding a good set of queries and relevant results for them is not an easy task. Even though there has been some work on evaluating queries in social networks, most notably by Bao et al. [6] who use DMOZ categories as global ground truth, such methods don't fit our user-centric search task. Here, it is not sufficient to consider global relevance measures, as the notion of relevance is highly subjective and dependent on the initiator of the query and her personal context. For example, a picture of a person may only be relevant if she is known to the query initiator. However, when we execute our queries in the context of a user taken from our collections, it is not possible to ask this user to assess the subjective relevance of a result item.

To solve this problem, we propose two independent evaluation methods, a *user-specific ground truth* and a *user study* with manual relevance assessments for selected queries. For a given query $Q(u, q_1 \dots q_n)$, we consider as user-specific ground truth the union of all documents which were tagged with all query tags by u or any of her direct friends. Our decision to consider also friend's documents as part of the ground truth is based on the fact that these are also documents that the user has direct access to and it is likely that the user has seen, and agreed with, the tags assigned. Since documents on the ground truth set contain tags from the query, we evaluate the queries on a residual collection where query tags by u and her friends are removed as they are known to lead towards relevant results. Note that this introduces a penalty for our method as the transitive friends with highest scores cannot contribute relevant results by definition. Queries for the ground truth were randomly selected from tag pairs with medium frequency in the corpus (between 1,000 and 2,000), the query initiator was chosen randomly among users that have previously used the query tags and

Without Tag Expansion											
α	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Flickr	0.39	0.42	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.41	0.36
LibraryThing	0.61	0.65	0.65	0.66	0.67	0.66	0.66	0.68	0.70	0.72	0.71
With Tag Expansion											
α	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Flickr	0.42	0.40	0.40	0.40	0.40	0.39	0.39	0.40	0.40	0.40	0.36
LibraryThing	0.61	0.63	0.64	0.65	0.65	0.65	0.65	0.67	0.69	0.72	0.71

Table 1: NDCG-10 for varying α, manual assessments

Without Tag Expansion											
α	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Delicious	0.15	0.25	0.27	0.33	0.34	0.35	0.35	0.37	0.39	0.39	0.36
LibraryThing	0.29	0.42	0.49	0.54	0.53	0.55	0.56	0.59	0.60	0.63	0.62
With Tag Expansion											
α	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Delicious	0.16	0.25	0.28	0.36	0.36	0.36	0.36	0.39	0.40	0.37	0.36
LibraryThing	0.30	0.41	0.46	0.53	0.52	0.53	0.55	0.57	0.59	0.61	0.60

Table 2: Precision@10 for varying α values, ground truth experiments

have at least one friend in the collection. This process yielded 150 queries for delicious and 184 queries for LibraryThing; note that this method cannot be applied to the Flickr data because there is almost no overlap in the pictures users tag.

For the user study on the LibraryThing data, we asked five colleagues to register with LibraryThing, tag at least 20 books there, and choose some friends among other users. They then suggested 28 queries related to the books they tagged. For Flickr, we collected 40 queries from other colleagues and randomly selected a (fictious) query initiator among the users in our Flickr crawl who used all query tags at least once on the same picture. We then ran the queries with different configurations of our algorithm and pooled the results. In the assessment phase, a volunteer (which was the query initiator for LibraryThing) was shown, in addition to the results for the query from the pool, the documents (i.e., pictures, bookmarks, or books) from the query initiator that contain at least one of the query tags, in order to understand the personal context of the query initiator. This way, we try to overcome the aforementioned problem of subjectively assessing result qualities with the eyes of the query initiator. The participant then marks each result as highly relevant, relevant, or nonrelevant in the context of the query initiator (without knowing which configuration generated the result).

6.2 Retrieval Effectiveness

We evaluate the effectiveness of our method by computing precision and normalized discounted cumulative gain (NDCG) [23] at certain cutoff levels, where relevant documents were determined either by human assessments or by the ground truth.

Results for the User Studies. The results from the user study are shown in Table 1. For both the Flickr and the LibraryThing data set the NDCG improved by increasing α, but dropped of when setting α to 1. This shows that while the semantic aspect is more important than social aspect for these specific datasets, the social component helps improving the search result, in particular for Flickr. This can also be seen with the precision@10, which for LibraryThing starts at 0.50 for $\alpha = 0.0$, increases to 0.53 for $\alpha = 0.9$ and drops to 0.40 for $\alpha = 1.0$.

Results for the User-Specific Ground Truth. The ground truth based experiments (Table 2 show very similar results. Again, result quality improves then increasing α, but drops when ignoring the social aspect and setting $\alpha = 1$. For Delicious the social aspect seems to be more important than for the other data sets, here the optimal value is $\alpha = 0.8$. Overall search effectiveness clearly benefits from integrating social scores.

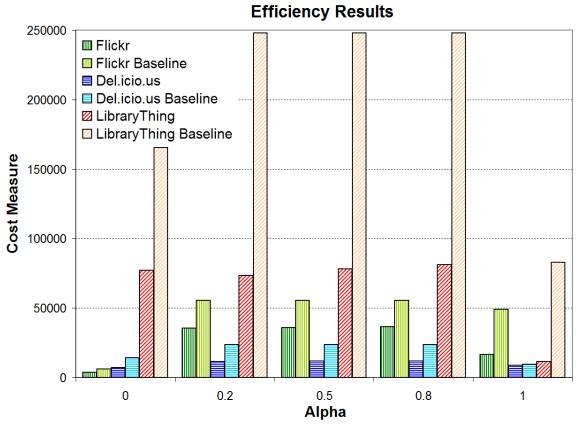

Figure 3: Average Execution Cost without tag expansion

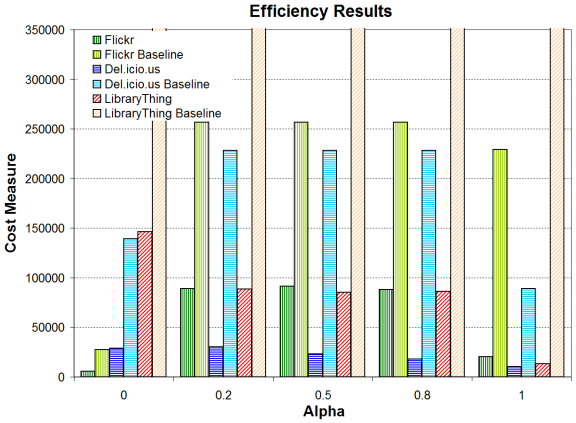

Figure 4: Average Execution Cost with tag expansion

6.3 Retrieval Efficiency

Our main concern in this paper has beeen the efficiency and scalability of the query processing. To assess the efficiency of the CONTEXTMERGE algorithm, we evaluated the two sets of queries used for the ground-truth based evaluation on the Librarything and delicious data sets, and in addition a set of 164 queries on the Flickr data set that were computed similarly to the others. The algorithm was implemented in Java, with the index lists stored in an Oracle 10g database. The experiments were done on an Windows-based server machine with four Opteron CPUs and 16GB of memory. We measured wall-clock runtimes and abstract cost in terms of disk accesses, where the cost for a random access was 100 and the cost for a sequential access 1. We compared our algorithm with a standard join-then-sort algorithm that reads all index lists involved in the query execution into memory, uses an in-memory hash join to combine entries for the same document, and finally does an in-memory sort of the candidate set to compute the top-k results.

For each collection, we performed a large variety of experiments to explore the space of alphas, thresholds for maximal tag expansion, and conjunctive vs. disjunctive evaluation. For space restrictions, we limit the discussion to selected results with conjunctive evaluation, results with disjunctive evaluation generally followed the same trends. Figure 3 depicts the average abstract cost per query for the three collections and selected values of α, evaluated with CONTEXTMERGE and the baseline without tag expansion. It is evident that our highly efficient top-k algorithm has at most half the cost of the baseline algorithm for most values of α, and shows even higher savings for the LibraryThing collection. Table 3 shows some additional details for the experiments on LibraryThing; it is

Configuration	time[s]	avg.#SA overall	avg.#SA to DOCS	avg.#SA to USERDOCS	avg.#RA overall	#avg. #exp
$alpha = 0.0$						
CONTEXTMERGE	0.70	70,588	0	70,588	65	0
CONTEXTMERGE w/ tag exp.	1.37	126,772	0	126,772	194	7
Baseline	1.43	165,352	0	165,352	0	0
Baseline w/ tag exp.	6.10	67,0405	0	67,0405	0	20
$alpha = 0.5$						
CONTEXTMERGE	0.68	76,012	21,834	54,178	23	0
CONTEXTMERGE w/ tag exp.	0.84	78,808	22,063	56,745	64	2
Baseline	2.0	248,093	82,742	165,351	0	0
Baseline w/ tag exp.	9.92	1,118,554	448,149	670,405	0	20
$alpha = 1.0$						
CONTEXTMERGE	0.14	11,341	11,341	0	1	0
CONTEXTMERGE w/ tag exp.	0.21	12,223	12,223	0	9	1
Baseline	0.59	82742	82,742	0	0	0
Baseline w/ tag exp.	3.43	448,149	448,149	0	0	20

Table 3: Efficiency details for LibraryThing with conjunctive evaluation

evident from the table that wall-clock runtime of our algorithm is also at least 50% better than that of the baseline (which is also the case for the other two collections).

Figure 4 shows abstract cost for the same setup, but now with tag expansion up to a limit of 10 similar tags. Note that the bars for the LibraryThing baseline experiment have been cut at 350,000. Here, the effectiveness of our dynamic tag expansion is clearly evident, as it saves factors of 3-5 compared to the baseline method which needs to completely scan the lists of all 10 related tags for each query tag. Table 3 again shows details for some experiments on LibraryThing; here, our highly efficient method manages to reduce runtime by up to an order of magnitude over the baseline. The column *avg.#exp* shows the average number of similar tags considered per query. Whereas the baseline methods needs to consider all tags, our self-throttling expansion technique requires only very few similar tags.

7. CONCLUSIONS

Social search is a promising direction to increase user-perceived query result quality. This paper developed an effective scoring model for user-centric searches in social networks, and introduced the CONTEXTMERGE algorithm to efficiently evaluate queries in such networks, dynamically including related users in the execution. Combining a top-k algorithm with dynamic tag expansion and dynamic expansion to similar users, CONTEXTMERGE is up to an order of magnitude faster in terms of measured runtime and cheaper in terms of abstract cost than the standard baseline method of processing inverted lists.

8. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[2] Y.-Y. Ahn et al. Analysis of topological characteristics of huge online social networking services. In *WWW*, 2007.

[3] S. Amer-Yahia et al. Challenges in searching online communities. *IEEE Data Eng. Bull.*, 30(2):23–31, 2007.

[4] V. N. Anh and A. Moffat. Pruned query evaluation using pre-computed impacts. In *SIGIR*, 2006.

[5] R. A. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *KDD*, 2007.

[6] S. Bao et al. Optimizing web search using social annotations. In *WWW*, 2007.

[7] H. Bast et al. Io-top-k: Index-access optimized top-k query processing. In *VLDB*, 2006.

[8] M. Bender et al. Peer-to-peer information search: Semantic, social, or spiritual? *IEEE Data Eng. Bull.*, 30(2):51–60, 2007.

[9] B. Billerbeck and J. Zobel. Questioning query expansion: An examination of behaviour and parameters. In *ADC*, 2004.

[10] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks*, 30(1–7):107–117, 1998.

[11] A. Damian et al. Peer-sensitive objectrank - valuing contextual information in social networks. In *WISE*, 2005.

[12] A. Das et al. Google news personalization: scalable online collaborative filtering. In *WWW*, 2007.

[13] P. A. Dmitriev et al. Using annotations in enterprise search. In *WWW*, 2006.

[14] M. Dubinko et al. Visualizing tags over time. *ACM Transactions on the Web*, 1(2), 2007.

[15] R. Fagin et al. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[16] S. Golder and B. A. Huberman. Usage patterns of collaborative tagging systems. *Journal of Information Science*, 32(2):198–208, April 2006.

[17] H. Halpin et al. The complex dynamics of collaborative tagging. In *WWW*, 2007.

[18] D. Heckerman et al. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, 2000.

[19] J. L. Herlocker et al. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1), 2004.

[20] P. Heymann et al. Can social bookmarking improve web search? In *WSDM*, 2008.

[21] P. Heymann and H. Garcia-Molina. Collaborative creation of communal hierarchical taxonomies in social tagging systems. Technical Report 2006-10, Stanford University, April 2006.

[22] A. Hotho et al. Information retrieval in folksonomies: Search and ranking. In *The Semantic Web: Research and Applications*, pages 411–426, 2006.

[23] K. Järvelin and J. Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *SIGIR*, 2000.

[24] R. Kumar et al. Structure and evolution of online social networks. In *KDD*, 2006.

[25] G. Linden et al. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1), 2003.

[26] A. Mislove et al. Exploiting social networks for internet search. In *HotNets*, 2006.

[27] J. Pouwelse et al. Tribler: A social-based peer-to-peer system. In *IPTPS*, 2006.

[28] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR*, 1994.

[29] B. M. Sarwar et al. Item-based collaborative filtering recommendation algorithms. In *WWW*, 2001.

[30] J. B. Schafer et al. Collaborative filtering recommender systems. In *The Adaptive Web*, 2007.

[31] C. Schmitz et al. Mining association rules in folksonomies. In *Data Science and Classification*. Springer, 2006.

[32] S. Sen et al. Tagging, communities, vocabulary, evolution. In *CSCW*, 2006.

[33] C. Tantipathananandh et al. A framework for community identification in dynamic social networks. In *KDD*, 2007.

[34] M. Theobald et al. Efficient and self-tuning incremental query expansion for top-k query processing. In *SIGIR*, 2005.

[35] S. Xu et al. Using social annotations to improve language model for information retrieval. In *CIKM*, 2007.

[36] J. Zhang et al. Expertise networks in online communities: structure and algorithms. In *WWW*, 2007.