As-You-Type Social Aware Search

Paul Lagrée, Bogdan Cautis, Hossein Vahabi

As-You-Type Social Aware Search

PAUL LAGREE, Université Paris-Sud, France
BOGDAN CAUTIS, Université Paris-Sud, France
HOSSEIN VAHABI, Pandora Media Inc., US

Modern search applications feature real-time as-you-type query search. In its elementary form, the problem consists in retrieving a set of k search results, i.e., performing a search with a given prefix, and showing the top ranked results. In this paper we focus on as-you-type keyword search over social media, that is data published by users who are interconnected through a social network. We adopt a "network-aware" interpretation for information relevance, by which information produced by users who are closer to the user issuing a request is considered more relevant. This query model raises new challenges for effectiveness and efficiency in online search, even when the intent of the user is fully specified, as a complete query given as input in one keystroke. This is mainly because it requires a joint exploration of the social space and traditional IR indexes such as inverted lists. We describe a memory-efficient and incremental prefix-based retrieval algorithm, which also exhibits an anytime behavior, allowing to output the most likely answer within any chosen running-time limit. We evaluate our approach through extensive experiments for several applications and search scenarios. We consider searching for posts in micro-blogging (Twitter and Tumblr), for businesses (Yelp), as well as for movies (Amazon) based on reviews. We also conduct a series of experiments comparing our algorithm with baselines using state-of-the-art techniques and measuring the improvements brought by several key optimizations. They show that our solution is effective in answering real-time as-you-type searches over social media.

1. INTRODUCTION

Web search is the main tool used today to access the enormous quantity of information available on the Web, and in particular in the social media. Starting from simple text-based search ranking algorithm, it is now an interdisciplinary topic involving data mining, machine learning, knowledge management, just to mention a few. Significant improvements have been done on how to answer keyword queries on the Web in the most effective way (e.g., by exploiting the Web structure, user and contextual models, user feedback, semantics, etc). However, answering information needs in social media applications (such as Tumblr, Twitter, or Facebook) often requires a significant departure from socially-agnostic approaches, which generally assume that the data being queried is decoupled from the users querying it.

While progress has been made in recent years to support this novel, social and *network-aware* query paradigm – especially towards efficiency and scalability – more remains to be done in order to address information needs in real applications. In particular, providing the most *accurate* answers while the user is typing her query, *almost instantaneously*, can be extremely beneficial, in order to enhance the user experience and to guide the retrieval process.

Figure 1 shows an example of as-you-type search in Tumblr. A user is typing a query as-you-type *"as yo"*. In the first part of the results (section "Search"), candidates are selected among queries within the query log and correspond to prefix based *query auto-completion* (such as *"as you are"*). In the second part though ("Blogs"), search results are presented for the partial query *"as yo"* (search result such as the blog *"love everybody"*). This suggestion framework is referred to as *as-you-type search* and is the focus of our work.

In this article, we extend as-you-type search – a functionality by now supported in most search applications, including Web search – to social data. In particular we extend existing algorithms for top-k retrieval (where k is the number of results returned, typically $k = 10$) over social data. Our solution, called TOPKS-ASYT (for TOP-k Social-aware search AS-You-Type), builds on the generic network-aware search approach of [Maniu and Cautis 2013b; Schenkel et al. 2008], and deals with three systemic changes:

(1) *Prefix matching:* answers must be computed following a query interpretation by which the last term in the query sequence can match tag / keyword prefixes.
(2) *Incremental computation:* answers must be computed *incrementally*, instead of starting a computation from scratch. For a query representing a sequence of terms $Q = [t_1, \ldots, t_r]$, we can

follow an approach that exploits what has already been computed in the query session so far, i.e., for the query $Q' = (t_1, \dots, t_{r-1}, t'_r)$, with t'_r being a one character shorter prefix of the term t_r.

(3) *Anytime output:* answers, albeit approximate, must be ready to be outputted at any time, and in particular after any given time lapse (e.g., $50 - 100ms$ is generally accepted as a reasonable latency for as-you-type search).

We consider a generic setting common to a plethora of social applications, where users produce unstructured content (keywords) in relation to items, an activity we simply refer to as *social tagging*. More precisely, our core application data can be modelled as follows: (i) users form a social network, which may represent relationships such as similarity, friendship, following, etc, (ii) items from a public pool of items (e.g., posts, tweets, videos, URLs, news, or even users) are "tagged" by users with keywords, through various interactions and data publishing scenarios, and (iii) users search for some k most relevant items by keywords.

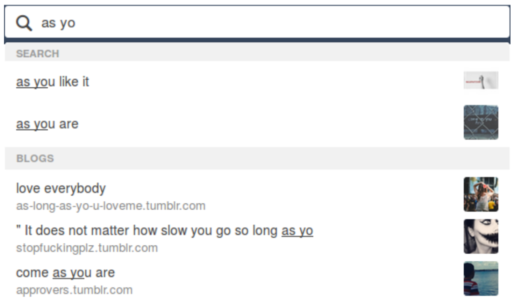
We devise a novel index structure for TOPKS-ASYT denoted CT-IL which is a combination of tries and inverted lists. While basic trie structures have been used in as-you-type search scenarios in the literature (e.g., see [Li et al. 2012] and the references therein), ranked access over inverted lists requires an approach that performs ranked completion more efficiently. Therefore, we rely on a trie structure tailored for the problem at hand, offering a good space-time tradeoff, namely the *completion trie* of [Hsu and Ottaviano 2013], which is an adaptation of the well-known Patricia trie using priority queues. This data structure is used as access layer over the inverted lists, allowing us to read in sorted order of relevance the possible keyword completions and the items for which they occur. Importantly, we use the completion trie also as a key internal component of our algorithm, in order to speed-up the incremental computation of results.

In this as-you-type search setting, it is necessary to serve in a short (fixed) lapse of time, after each keystroke and in social-aware manner, top-k results matching the query in its current form, i.e., the terms $t_1, \dots, t_{r-1}$, and all possible completions of the term t_r. This must be ensured independently of the execution configuration, data features, or scale. This is why we ensure that our algorithms have also an *anytime* behaviour, being able to output the most likely result based on all the preliminary information obtained until a given time limit for the TOPKS-ASYT run is reached.

1.1. Comparison with previous publication

We extend in this article a preliminary study published in [Lagrée et al. 2015], introducing the following novel contributions, which allow us to give a complete picture on our algorithmic solutions:

(1) *Supernodes optimization:* We introduce and evaluate experimentally a novel feature, denoted *supernodes*, which consists in clustering users in groups of chosen size in order to speed up graph exploration. The goal is to improve the precision of TOPKS-ASYT when the time allocated to serve responses is greatly constrained.
(2) *Baselines:* We describe three baseline methods using state-of-the-art algorithms and we evaluate the gains of TOPKS-ASYT with respect to them.
(3) *SimRank proximity:* We also consider a path-based similarity measure (as opposed to the neighborhood-based one used before), namely SimRank, as the proximity indicator in the social network, and we evaluate the performance of our algorithm under this model assumption.
(4) *Model for multi-word queries:* When dealing with multi-word queries, we noticed that the last term (seen as a potential prefix in the algorithm) yields consistently higher scores compared with those of the terms preceding it. We propose a calibration of the prefix score to smooth this effect and have each query term's score contribution (including the prefix one) similar.
(5) *New datasets:* We add experiments based on data from an entirely different social media, the one of movie reviews in Amazon, which allows us to verify the robustness of TOPKS-ASYT to keyword search with medium-size tagging contributions from users. This new dataset completes the previous pool composed of Twitter and Tumblr, two of the most popular micro-blogging platforms today, and Yelp, a website where users write reviews on local businesses.



as yo

SEARCH

as you like it

as you are

BLOGS

love everybody
as-long-as-yo-u-loveme.tumblr.com

" It does not matter how slow you go so long as yo
stopfuckingplz.tumblr.com

come as you are
approvers.tumblr.com

Fig. 1. An as-you-type search example (“Search”: autocompletion, “Blogs”: as-you-type results).

(6) *New experiments*: Besides new experiments concerning the novel aspects (supernodes, baselines, SimRank proximity), we also add new experiments to assess (a) the benefit of the incremental computation, and (b) the performance of our algorithm for *multi-word queries*.
(7) *Analysis:* We characterize the computational complexity for the main data structures and algorithmic steps of our method.

The paper is organised as follows. In Section 2 we discuss the main related works. We lay out our data and query model in Section 3. Our technical contribution is described in Section 4 and is evaluated experimentally in Section 5. We conclude and discuss follow-up research in Section 6.

2. RELATED WORK

Top-k retrieval algorithms, such as the Threshold Algorithm (TA) and the No Random Access algorithm (NRA) [Fagin et al. 2001], which are *early-termination* and exploit the textual similarity, have been adapted to network-aware query models for social applications, following the idea of biasing results by social relevance [Yahia et al. 2008; Schenkel et al. 2008; Maniu and Cautis 2013b] and even time freshness [Li et al. 2015]. For more details on personalized search in social media we refer the interested readers to the references therein.

As-you-type search (also known as type-ahead search) and query auto-completion are two of the most important features in search engines today, and belong to the broader area of *instant search* (see [Venkataraman et al. 2016] for a recent tutorial on the topic). They can be seen as facets of the same paradigm: providing accurate feedback to queries on-the-fly, i.e., as they are being typed (possibly with each keystroke). In as-you-type search, feedback comes in the form of the most relevant answers for the query typed so far, allowing some terms (usually, the last one in the query sequence) to be prefix-matched. In query auto-completion, a list of the most relevant query candidates is to be shown for selection, possibly with results for them. We discuss each of these two directions separately. Also, instant search shares many challenges with *exploratory search*, for settings dealing with under-specified, undirected, and even interactive search tasks (see [Ahukorala et al. 2015] and the references therein).

The problem we study in this paper, namely top-k as-you-type search for multiple keywords, has been considered recently in [Li et al. 2012], which mainly differs from our work in the absence of social dimension in data. There, the authors consider various adaptations of the well-known TA/NRA top-k algorithms of [Fagin et al. 2001], even in the presence of minor typing errors (fuzzy search), based on standard tries. A similar fuzzy interpretation for full-text search was followed in [Ji et al. 2009], yet not in a top-k setting. The techniques of [Li et al. 2010] rely on precomputed materialization of top-k results, for values of k known in advance. In [Bast et al. 2008; Bast and Weber 2006], the goal is finding all the query completions leading to results as well as listing these results, based on inverted list and suffix array adaptations; however, the search requires a full computation and then ranking of the results. For structured data instead of full text, type-ahead search has been considered in [Feng and Li 2012] (XML) and in [Li et al. 2013] (relational data). Finally, [Zhong et al. 2012] study location-aware as-you-type search by providing location-biased answers, instead of socially-biased ones.

Query auto-completion is the second main direction for instant response to queries in the typing, by which some top query completions are presented to the user (see for example [Zhang et al. 2015; Shokouhi and Radinsky 2012; Shokouhi 2013; Cai et al. 2014] and the references therein). This is done either by following a predictive approach, or by pre-computing completion candidates and storing them in trie structures. Probably the best known example today is the one of Google's instant search, which provides both query predictions (in the search box) and results for the top prediction. In [Fafalios and Tzitzikas 2015], the authors discuss in depth various systems choices involving index partitioning or caching, for query auto-completion under typo-tolerant and word-order tolerant assumptions. Query suggestion goes one step further by proposing alternative queries, which are not necessarily completions of the input one (see for instance [Vahabi et al. 2013; Jiang et al. 2014]). In comparison, our work does not focus on queries as first-class citizens, but on instant results to incomplete queries.

Person search represents another facet of "social search", related to this paper, as the task of finding highly relevant persons for a given seeker and keywords. Usually, the approach used in this type of application is to identify the most relevant users, and then to filter them by the query keywords [Potamias et al. 2009; Bahmani and Goel 2012]. In this area, [Curtiss et al. 2013] describes the main aspects of the Unicorn system for search over the Facebook graph, including a typeahead feature for user search. A similar search problem, finding a sub-graph of the social network that connects two or more persons, is considered under the instant search paradigm in [Wu et al. 2012].

Several space-efficient trie data structures for ranked (top-k) completion have been studied recently in [Hsu and Ottaviano 2013], offering various space-time tradeoffs, and we rely in this paper on one of them, namely the completion trie. In the same spirit, data structures for the more general problem of substring matching for top-k retrieval have been considered in [Hon et al. 2009].

3. MODEL

We adopt in this paper a well-known generic model of social relevance for information, previously considered among others in [Maniu and Cautis 2012; 2013b; Yahia et al. 2008; Schenkel et al. 2008]. In short, the social bias in scores reflects the social proximity of the producers of content with respect to the seeker (the user issuing a search query), where proximity is obtained by some aggregation of shortest paths (in the social space) from the seeker towards relevant pieces of information.

3.1. Notations and context

We consider a social setting, in which we have a set of items (could be text documents, blog posts, tweets, URLs, photos, etc) $\mathcal{I} = \{i_1, \ldots, i_m\}$, each tagged with one or more distinct tags from a tagging vocabulary $\mathcal{T} = \{t_1, t_2, \ldots, t_l\}$, by users from $\mathcal{U} = \{u_1, \ldots, u_n\}$. We denote our set of unique triples by $Tagged(v, i, t)$, each such triple saying that a user v tagged the item i with tag t. $Tagged$ encodes many-to-many relationships: in particular, any given item can be tagged by multiple users , and any given user can tag multiple items. We also assume that a user will tag a given item with a given tag at most once.

While social media applications can adopt for their explicit social links either the directed graph model (e.g. , Twitter or Tumblr) or the undirected one (e.g., Yelp or Facebook), we assume in the following that users form a social *similarity* network, modeled for our purposes as an undirected weighted graph $G = (\mathcal{U}, E, \sigma)$, where nodes are users and the σ function associates to each edge $e = (u_1, u_2)$ a value in $(0, 1]$, called *the proximity* (social) score between u_1 and u_2. Proximity may come either from explicit social signals (e.g., friendship links, follower/followee links), or from implicit social signals (e.g., tagging similarity), or from combinations thereof.[1]

To illustrate, one σ instantiation, i.e., similarity measure, we rely on in our experiments is the *Dice's coefficient:* given two users u and v, we compare their friends (respectively vocabulary,

[1]A key model and experimental choice in our paper is to define the social dimension by implicit links, using similarity-based (thus reciprocal) proximity metrics. This choice is orthogonal to whether we get directed or undirected edges and our techniques extend easily to settings in which the input social dimension has directed edges instead of the undirected ones.

items) to compute a local social (respectively tag, item) similarity. For example, denoting by N_u and N_v the set of users connected to u and v, the Dice's social coefficient is computed as follows:

$$\sigma^{Dice}(u,v) = \frac{2|N_u \cup N_v|}{|N_u| + |N_v|} \tag{1}$$

3.2. General keyword search problem

The general (not as-you-type) keyword search can be formulated as follows: given a seeker user s, a keyword query $Q = (t_1, \ldots, t_r)$ (a set of r distinct terms/keywords) and a result size k, the top-k keyword search problem is to compute the (possibly ranked) list of the k items having the highest scores with respect to s and the query Q. We describe hereafter the model ingredients on which we rely to identify query results in a social media context, and then we present in Section 3.4 the particular search problem instance we study in this paper.

3.3. Social and textual relevance

We model by $score(i \mid s,t)$, for a seeker s, an item i, and a tag t, the relevance of that item for the given seeker and query term t. Generally, we assume

$$score(i \mid s,t) = h(fr(i \mid s,t)), \tag{2}$$

where $fr(i \mid s,t)$ is the *frequency* of item i for seeker s and tag t, and h is a positive monotone function (e.g., could be based on inverse term frequency, BM25, etc).

Given a query $Q = (t_1, \ldots, t_r)$, the overall score of i for seeker s and Q is simply obtained by summing the per-tag scores:

$$score(i \mid s,Q) = \sum_{t_j \in Q} score(i \mid s,t_j). \tag{3}$$

(Note that this naturally corresponds to an OR semantics, where items that do not necessarily match all the query tags may still be selected; for an AND one, each term's score should be non-empty.)

Social relevance model. In an exclusively social interpretation, we can explicitate the $fr(i \mid s,t)$ measure by the *social frequency* for seeker s, item i, and one tag t, denoted $sf(i \mid s,t)$. This measure adapts the classic term frequency (tf) measure to account for the seeker and its social proximity to relevant taggers. We consider that each tagger brings her own weight (proximity) to an item's score, and we define social frequency as follows:

$$sf(i \mid s,t) = \sum_{v \in \{v \mid Tagged(v,i,t))\}} \sigma(s,v). \tag{4}$$

Note that, under the frequency definition of Eq. (2), we would follow a ranking approach by which information that may match the query terms but does not score on the social dimension (i.e., is disconnected from the seeker) is deemed entirely irrelevant.

Network-aware relevance model. A more generic relevance model, which does not solely depend on social proximity but is *network-aware*, is one that takes into account textual relevance scores as well. For this, we denote by $tf(t,i)$ the *term frequency* of t in i, i.e., the number of times i was tagged with t, and $IL(t)$ is the inverted list of items for term t, ordered by term frequency.

The frequency score $fr(i \mid s,t)$ is defined as a linear combination of the previously described social relevance and the textual score, with $\alpha \in [0,1]$, as follows:

$$fr(i \mid s,t) = \alpha \times tf(t,i) + (1-\alpha) \times sf(i \mid s,t). \tag{5}$$

(This formula combines a global popularity of the item with one among people close to the seeker.)

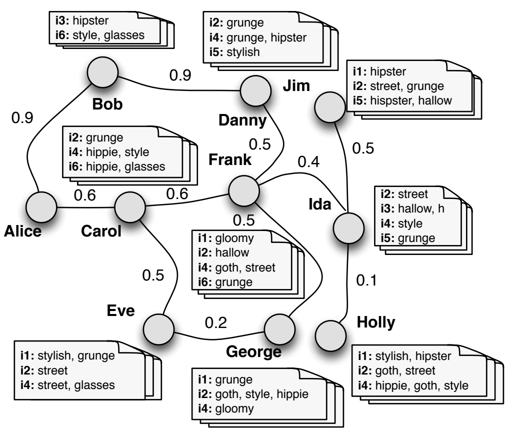


Fig. 2. Running example: social proximity and tagging.

Remark. We believe that this simple model of triples for social data is the right abstraction for quite diverse types of social media. Consider Tumblr [Chang et al. 2014]: one broadcasts posts to followers and rebroadcasts incoming posts; when doing so, the re-post is often tagged with chosen tags or short descriptions (hashtags). We can thus see a post and all its re-posted instances as representing one informational item, which may be tagged with various tags by the users broadcasting it. Text appearing in a blog post can also be interpreted as tags, provided either by the original author or by those who modified it during subsequent re-posts; it can also be exploited to uncover implicit tags, based on the co-occurrence of tags and keywords in text. Furthermore, a post that is clicked-on in response to a Tumblr search query can be seen as being effectively tagged (relevant) for that query's terms. All this data has obviously a social nature: e.g., besides existing follower/followee links, one can even use similarity-based links as social proximity indicators.

Example 3.1. We depict in Figure 2 a social network and the tagging activity of its users, for a running example based on popular tags from the fashion domain in Tumblr. There, for seeker Alice, we have for instance, for $\alpha = 0.2$, $tf(glasses, i6) = 2$, and then

$$\begin{aligned} sf(i6 \mid Alice, glasses) &= \sigma(Alice, Bob) + \sigma(Alice, Carol) = 0.9 + 0.6 = 1.5, \\ fr(i6 \mid Alice, glasses) &= 0.8 \times 1.5 + 0.2 \times 2. \end{aligned}$$

Extended proximity. The model described so far takes into account only the immediate neighbourhood of the seeker (the users it connects to explicitly). In order to broaden the scope of the query and go beyond one's vicinity in the social network, we also account for users that are indirectly connected to the seeker, following a natural interpretation that user links and the query relevance they induce are (at least to some extent) transitive. To this end, we denote by σ^+ the resulting measure of *extended proximity*, which is to be computed from σ for any pair of users connected by at least one path in the network. Now, σ^+ can replace σ in the definition of social frequency Eq. (4).

For example, one natural way of obtaining extended proximity scores is by (i) multiplying the weights on a given path between the two users, and (ii) choosing the maximum value over all the possible paths. Another possible definition for σ^+ can rely on an aggregation that penalizes long paths, via an *exponential decay* factor, in the style of the Katz social proximity [Katz 1953]. More generally, any aggregation function that is monotonically non-increasing over a path, can be used here. Under this monotonicity assumption, one can browse the network of users *on-the-fly* (at query time) and "sequentially", i.e., visiting them in the order of their proximity with the seeker.

Hereafter, when we talk about proximity, we refer to the extended one, and, for a given seeker s, the *proximity vector* of s is the list of users with non-zero proximity with respect to it, ordered decreasingly by proximity values (we stress that this vector is not necessarily known in advance).

Example 3.2. For example, for seeker Alice, when extended proximity between two users is defined as the maximal product of scores over paths linking them, the users ranked by proximity w.r.t. Alice are in order $Bob : 0.9, Danny : 0.81, Carol : 0.6, Frank : 0.4, Eve : 0.3, George : 0.2, Ida : 0.16, Jim : 0.07, Holly : 0.01$.

3.4. The as-you-type search problem

With respect to the general keyword search problem formulated before, we consider in this paper a specialized and potentially more useful level of search service for practical purposes, in which queries are being answered *as they are typed*. Instead of assuming that the query terms are given all at once, a more realistic assumption is that input queries are sequences of terms $Q = [t_1, \ldots, t_r]$, in which all terms but the last are to be matched exactly, whereas the last term t_r is to be interpreted as a tag potentially still in the writing, hence matched as a tag prefix.

We extend the query model in order to deal with tag prefixes p by defining an item's score for p as the maximal one over all possible completions of p:

$$sf(i \mid s, p) = \max_{t \in \{p's\ completions\}} sf(i \mid s, t) \tag{6}$$

$$tf(p, i) = \max_{t \in \{p's\ completions\}} tf(t, i) \tag{7}$$

(Note that when we compute the importance of an item, we might consider two different tag completions, for the social contribution and for the popularity one.)

Example 3.3. If Alice's query is `hipster g`, as `g` matches the tags `gloomy`, `glasses`, `goth` and `grunge`, we have $sf(i4 \mid Alice, \mathtt{g})$ as

$$\begin{aligned} & \max_{t \in \{\mathtt{g}\ completions\}} sf(i4 \mid Alice, t) \\ = \ & \max[sf(i4 \mid Alice, \mathtt{gloomy}), sf(i4 \mid Alice, \mathtt{glasses}), sf(i4 \mid Alice, \mathtt{grunge}), sf(i4 \mid Alice, \mathtt{goth})] \\ = \ & \max[0.2, 0.3, 0.81, 0.41] = 0.81 \end{aligned}$$

4. AS-YOU-TYPE SEARCH ALGORITHMS

We revisit here the network-aware retrieval approach of [Maniu and Cautis 2013b; Schenkel et al. 2008], which belongs to the family of early termination top-k algorithms known as *threshold algorithms*, of which [Fagin et al. 2001]'s TA (the Threshold Algorithm) and NRA (No Random-access Algorithm) are well-known examples.

In the social-aware retrieval setting, when social proximity determines relevance, the data exploration must jointly consider the network (starting from the seeker and visiting users in descending proximity order), the per-user/personal tagging spaces, and all available socially-agnostic index structures such as inverted lists. It is thus important for efficiency to explore the social network by order of relevance/proximity to the seeker, as to access all the necessary index structures, in a *sequential manner* as much as possible. We favor such an approach here, instead of an *incomplete* "one dimension at a time" one, which would first rely on one dimension to identify a set of candidate items, and then use the scores for the other dimension to re-rank or filter out some of the candidates.

4.1. Non-incremental algorithm

We first describe the TOPKS-ASYT approach for exclusively social relevance ($\alpha = 0$) and without incremental computation, namely when the full sequence of terms is given in one keystroke, with the last term possibly a prefix, as $Q = (t_1, \ldots, t_r)$. We follow an early-termination approach that is "user-at-a-time": its main loop step visits a new user and the items that were tagged by her with query terms. Algorithm 1 gives the flow of TOPKS-ASYT.

Main inputs. For each user u and tag t, we assume a precomputed selection over the *Tagged* relation, giving the items tagged by u with t; we call these the *personal spaces* (in short, p-spaces). No particular order is assumed for the items appearing in a user list.

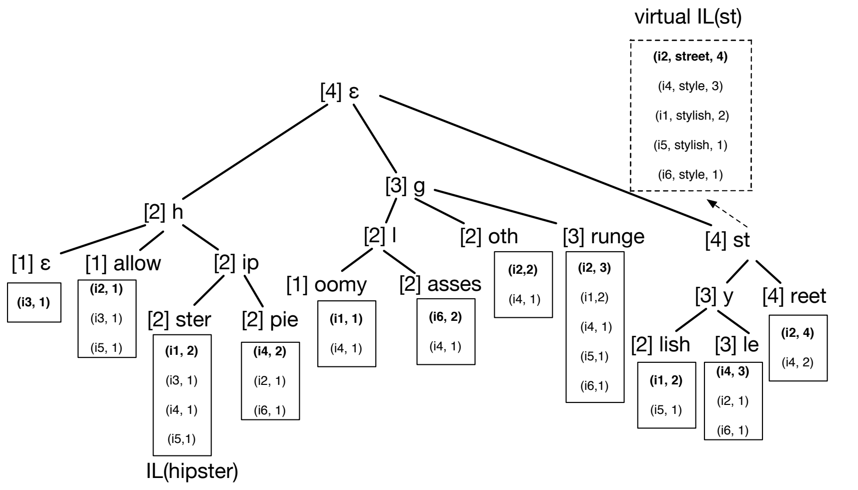


Fig. 3. The CT-IL index.

We also 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 $tf(t, i)$[2], ordered descending by them. The lists can be seen as unpersonalized indexes. A completion trie over the set of tags represents the access layer to these lists. As in Patricia tries, a node can represent more than one character, and the scores corresponding to the heads of the lists are used for ranked completion: each leaf has the score of the current entry in the corresponding inverted list, and each internal node has the maximal score over its children (see example below). This index structure is denoted hereafter the CT-IL index.

Example 4.1 (*CT-IL index*). We give in Figure 3 an illustration of the main components of CT-IL, for our running example. Each of the tags has below it the inverted list (the one of the `hippie` tag is explicitly indicated). The cursor positions in the lists are in bold. By storing the maximal score at each node (in brackets in Figure 3), the best (scoring) completions of a given prefix can be found by using a priority queue, which is initialized with the highest node matching that prefix. With each pop operation, either we get a completion of the prefix, or we advance towards one, and we insert in the queue the children of the popped node.

For comparison, we also illustrate in Figure 4 the CT-IL index that would allow us to process efficiently Alice's top-k queries, without the need to resort to accesses in social network and p-spaces. Obviously, building such an index for each potential seeker would not be feasible.

While leaf nodes in the trie correspond to concrete inverted lists, we can see each internal node of the trie and the corresponding keyword prefix as described by a "virtual inverted list", i.e., the ranked union of all inverted lists below that node. As defined in Eq. (6-7), for such a union, for an item appearing in entries of several of the unioned lists, we keep only the highest-scoring entry. In particular, for the query term t_r, by $IL(t_r)$ we refer to the virtual inverted list corresponding to this prefix. There is a notable difference between the concrete inverted lists and the virtual ones: in the former, entries can be seen (stored) as pairs $(item, score)$ (the tag is implied); in the latter, entries must be the form $(item, tag, score)$, as different tags (completions) may appear in such a list.

For each $t \in \{t_1, \dots, t_r\}$, we denote by $top\_item(t)$ the item present at the current (unconsumed) position of $IL(t)$, we use $top\_tf(t)$ as short notation for the term frequency associated with this item, and, for $IL(t_r)$, we also denote by $top\_tag(t_r)$ the t_r completion in the current entry.

Example 4.2 (*Virtual lists*). The virtual inverted list for the prefix `st` is given in Fig. 3. The $top\_tag($`st`$)$ is `street`, for $top\_item($`st`$)$ being $i2$, for its entry scored 4 dominates the one scored

[2]Even when $\alpha = 0$, although social frequency does not depend directly on tf scores, we exploit the inverted lists and the tf scores by which they are ordered, to better estimate score bounds.

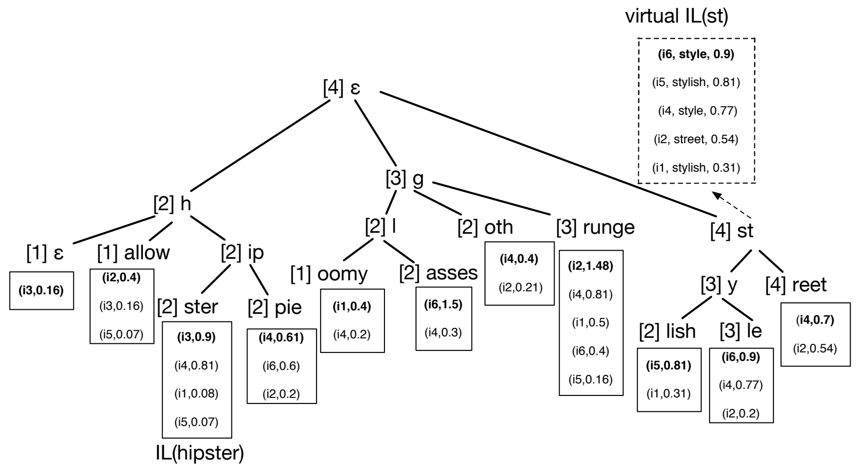


Fig. 4. Alice's personalized CT-IL index.

only 2, hence with a $top\_tf(\texttt{st})$ of 4. A similar one, for the "personalized" CT-IL index for seeker Alice is given in Fig. 4.

Candidate buffers. For each tag $t \in \{t_1, \ldots, t_{r-1}\}$, we keep a list D_t of candidate items i, along with a sound score range: a lower-bound and an upper-bound for $sf(i \mid s, t)$ (to be explained hereafter). Similarly, in the case of t_r, for each completion t of t_r already encountered during the query execution in p-spaces (i.e., by triples (u, i, t) read in some u's p-space), we record in a D_t list the candidate items and their score ranges. Candidates in these D-buffers are sorted in descending order by their score lower-bounds.

An item becomes candidate and is included in D-buffers only when it is first met in a $Tagged$ triple matching a query term.

For uniformity of treatment, a special item $*$ denotes all the yet unseen items, and it implicitly appears in each of the D-lists; note that, in a given D_t buffer, $*$ represents both items which are not yet candidates, but also candidate items which may already be candidates but appear only in other D-buffers (for tags other than t).

Main algorithmic components. When accessing the CT-IL index, inverted list entries are consumed in some $IL(t)$ only when the items they refer to are candidates (they appear in at least one $D_{t'}$ buffer, which may not be D_t itself)[3]. We keep in lists called CIL_t (for *consumed IL entries*) the items read (known candidates) in the inverted lists (virtual or concrete), for t being either in $\{t_1, \ldots, t_{r-1}\}$ or a completion of t_r for which a triple $(item, t, score)$ was read in the virtual list of t_r. We also keep by the set C all t_r completions encountered so far in p-spaces. Note that t_r completions encountered in p-spaces may not necessarily coincide with those encountered in $IL(t_r)$.

For each t being either in $\{t_1, \ldots, t_{r-1}\}$ or a completion of t_r already in C, by $unseen\_users(i, t)$ we denote the maximal number of yet unvisited users who may have tagged item i with tag t. This number is initially set to the maximal possible term frequency of t over all items. $unseen\_users(i, t)$ then reflects at any moment during the run of the algorithm the difference between the number of taggers of i with t already visited and one of either

— the value $tf(t, i)$, if this term frequency has been read already by accessing CT-IL, or otherwise
— the value $top\_tf(t)$, if $t \in \{t_1, \ldots, t_{r-1}\}$, or
— the value $top\_tf(t_r)$, if t is instead a completion of t_r.

[3]The rationale is that our algorithm does not make any "wild guesses", avoiding reads that may prove to be irrelevant and thus leading to sub-optimal performance.

ALGORITHM 1: TOPKS-ASYT (non-incremental, for $\alpha = 0$)

```
Input: seeker $s$, query $Q = (t_1, \ldots, t_r)$
forall the users $u$ do
    $\sigma^+(s,u) = -\infty$;
end
forall the tags $t \in \{t_1, \ldots, t_{r-1}\}$ do
    $sf(i \mid s,t) = 0$; $D_t = \emptyset$, $CIL_t = \emptyset$;
    set $IL(t)$ position on first entry;
end
set $IL(t_r)$ position on first entry, $\sigma^+(s,s) = 0$, $C = \emptyset$ ($t_r$ completions);
$H \leftarrow$ priority queue on users; init. $\{s\}$, computed on-the-fly;
while $H \neq \emptyset$ do
    u=EXTRACT_MAX(H);
    PROCESS_P_SPACE(u);
    PROCESS_CT-IL;
    if termination condition then
        break;
    end
end
return top-$k$ items;
```

For known candidates i of some D_t, we accumulate in $sf(i \mid s,t)$ the social score (initially 0). Each time we visit a user u having a triple (u,i,t) in her p-space (Algorithm 2), we can

(1) update $sf(i \mid s,t)$ by adding $\sigma^+(s,u)$ to it, and
(2) decrement $unseen\_users(i,t)$; when this reaches 0, the social frequency $sf(i \mid s,t)$ is final.

The maximal proximity score of yet to be visited users is denoted $max\_proximity$. With this proximity bound, a sound score range for candidates i in D_t buffers is computed and maintained as

— a score upper-bound (maximal score) MAXSCORE$(i \mid s,t)$, by $max\_proximity \times unseen\_users(i,t) + sf(i \mid s,t)$.
— a score lower-bound (minimal score), MINSCORE$(i \mid s,t)$, by assuming that the current social frequency $sf(i \mid s,t)$ is the final one (put otherwise, all remaining taggers u of i with t, which are yet to be encountered, have $\sigma^+(s,u) = 0$).

Consuming the inverted list entries (Algorithm 3) in CT-IL, whenever top items become candidates, allows us to keep as accurate as possible the worst-case estimation on the number of unseen taggers. When such a tuple $(i, t, score)$ is accessed, we can do adjustments on score estimates:

(1) if $i \in D_t$, we can mark the number of unseen taggers of i with t as no longer an estimate but an *exact* value; from this point on, the number of unseen users will only change whenever new users who tagged i with t are visited,
(2) by advancing to the next best item in $IL(t)$, for $t \in \{t_1, \ldots, t_{r-1}\}$, we can refine the $unseen\_users(i', t)$ estimates for all candidate items i' for which the exact number of users who tagged them with t is yet unknown,
(3) by advancing to the next best item in $IL(t_r)$, with some $t = top\_tag(t_r)$ completion of t_r, if $t \in C$, we can refine the estimates $unseen\_users(i', t)$ for all candidate items $i' \in D_t$ for which the exact number of users who tagged them with t is yet unknown.

Termination condition From the per-tag D_t buffers, we can infer lower-bounds on the global score w.r.t. Q for a candidate item (as defined in Eq. (3)) by summing up its score lower-bounds from $D_{t_1}, \ldots, D_{t_{r-1}}$ and its maximal score lower-bound across all D_t lists, for completions t of t_r. Similarly, we can infer an upper-bound on the global score w.r.t. Q by summing up score upper-bounds from $D_{t_1}, \ldots, D_{t_{r-1}}$ and the maximal upper-bound across all D_t lists, for completions t.

ALGORITHM 2: SUBROUTINE PROCESS_P_SPACE(u)

```
forall the tags $t \in \{t_1, \ldots, t_{r-1}\}$, triples $Tagged(u, i, t)$ do
    if $i \notin D_t$ then
        add $i$ to $D_t$, $sf(i \mid s, t) \leftarrow 0$;
        $unseen\_users(i, t) \leftarrow top\_tf(t)$;
    end
    $unseen\_users(i, t) \leftarrow unseen\_users(i, t) - 1$;
    $sf(i \mid s, t) \leftarrow sf(i \mid s, t) + \sigma^+(s, u)$;
end
forall the tags $t$ completions of $t_r$, triples $Tagged(u, i, t)$ do
    if $t \notin C$ then
        add $t$ to $C$, $D_t = \emptyset$ ;
    end
    if $i \notin D_t$ then
        add $i$ to $D_t$, $sf(i \mid s, t) \leftarrow 0$;
        $unseen\_users(i, t) \leftarrow top\_tf(t)$;
    end
    $unseen\_users(i, t) \leftarrow unseen\_users(i, t) - 1$;
    $sf(i \mid s, t) \leftarrow sf(i \mid s, t) + \sigma^+(s, u)$;
end
```

After sorting the candidate items (the wildcard item included) by their global score lower-bounds, TOPKS-ASYT can terminate whenever (i) the wildcard item is not among the top-k ones, and (ii) the score upper-bounds of items not among the top-k ones are less than the score lower-bound of the kth item in this ordering (we know that the top-k can no longer change).

As in [Maniu and Cautis 2013b], it can be shown that TOPKS-ASYT visits users who may be relevant for the query in decreasing proximity order and, importantly, that it visits as few users as possible (it is *instance optimal* for this aspect, in the case of exclusively social relevance).

Example 4.3. Revisiting our running example, let us assume Alice requires the top-2 items for the query $Q = (\texttt{style}, \texttt{gl})$ $(\alpha = 0)$. The first data access steps of TOPKS-ASYT are as follows: at the first execution of the main loop step, we visit Bob, get his p-space, adding $i6$ both to the $D_{\texttt{style}}$ buffer and to a $D_{\texttt{glasses}}$ one. There may be at most two other taggers of $i6$ with `style` ($unseen\_users(i6, \texttt{style})$), and at most one other tagger of $i6$ with `glasses` ($unseen\_users(i6, \texttt{glasses})$). No reading is done in $IL(\texttt{style})$, as its current entry gives the non-candidate item $i4$, but we can advance with one pop in the virtual list of the `gl` prefix, for candidate item $i6$. This clarifies that there is *exactly* one other tagger with `glasses` for $i6$. After this read in the virtual list of `gl`, we have $top\_item(\texttt{gl}) = i1$ (if we assume that items are also ordered by their ids). At this point $max\_proximity$ is 0.81. Therefore, we have

$$\text{MAXSCORE}(i6 \mid Alice, \texttt{style}) = 0.81 \times 2 + 0.9, \text{MINSCORE}(i6 \mid Alice, \texttt{style}) = 0.9$$
$$\text{MAXSCORE}(i6 \mid Alice, \texttt{glasses}) = 0.81 \times 1 + 0.9, \text{MINSCORE}(i6 \mid Alice, \texttt{glasses}) = 0.9$$

We thus have that $score(i6|Alice, Q)$ is between 1.8 and 4.23.

At the second execution of the main loop step, we visit $Danny$, whose p-space does not contain relevant items for Q. But a side-effect of this step is that $max\_proximity$ becomes 0.6, affecting the upper-bound scores above: $score(i6 \mid Alice, Q)$ can now be estimated between 1.8 and 3.6.

At the third execution of the main loop step, we visit $Carol$, and find the relevant p-space entries for $i4$ (with tag `style`) and $i6$ (with tag `glasses`). Now $max\_proximity$ becomes 0.4. Also, we can advance with one pop in the inverted list of `style`. This clarifies that there are *exactly* 2 other taggers with `style` on $i4$, and now we have $top\_item(\texttt{gl}) = i1$ and $top\_item(\texttt{style}) = 2$. This makes $score(i6 \mid Alice, Q)$ to be known precisely at 2.4, $score(i4 \mid Alice, Q)$ to be estimated between 0.6 and $0.6 + 3 \times 0.4 = 1.8$, and $score(* \mid Alice, Q)$ is at most 0.8. At this point the algorithm has reached the termination condition.

ALGORITHM 3: SUBROUTINE PROCESS_CT-IL

```
while $\exists t \in Q$ s.t. $i = top\_item(t) \in \bigcup_x D_x$ do
    if $t \neq t_r$ then
        $tf(t,i) \leftarrow top\_tf(t)$ ($t$'s frequency in $i$ is now known);
        advance $IL(t)$ one position;
        $\Delta \leftarrow tf(t,i) - top\_tf(t)$ (the top_tf drop);
        add $i$ to $CIL_t$;
        forall the items $i' \in D_t \setminus CIL_t$ do
            $unseen\_users(i',t) \leftarrow unseen\_users(i',t) - \Delta$;
        end
    end
    if $t = t_r$ then
        $t' \leftarrow top\_tag(t_r)$ (some $t_r$ completion $t'$);
        $tf(t',i) \leftarrow top\_tf(t_r)$ ($t'$'s frequency in $i$ known);
        advance $IL(t_r)$ one position;
        $\Delta \leftarrow tf(t',i) - top\_tf(t_r)$ (the top_tf drop);
        add $i$ to $CIL_{t'}$ or set $CIL_{t'}$ to $\{i\}$ if previously empty;
        forall the $t'' \in C$ and items $i' \in D_{t''} \setminus CIL_{t''}$ do
            $unseen\_users(i',t'') \leftarrow unseen\_users(i',t'') - \Delta$;
        end
    end
end
```

4.2. Adaptations for the network-aware case

We sketch in this section the necessary extensions to Algorithm 1 for arbitrary α values, hence for any textual-social relevance balance. When $\alpha \in [0,1]$, at each iteration, the algorithm can alternate between two possible execution branches: the *social branch* (the one detailed in Algorithm 1) and a *textual branch*, which is a direct adaptation of NRA over the CT-IL structure, reading in parallel in all the query term lists (concrete or virtual). Now, items can become candidates even without being encountered in p-spaces, when read in inverted lists during an execution of the textual branch. As before, each read from CT-IL is associated with updates on score estimates such as $unseen\_users$. For a given item i and tag t, the maximal possible fr-score can be obtained by adding to the previously seen maximal possible sf-score (weighted now by $1-\alpha$) the maximal possible value of $tf(t,i)$; the latter may be known (if read in CT-IL), or estimated as $top\_tf(t)$ otherwise. Symmetrically, the minimal possible value for $tf(t,i)$ is used for lower bounds; if not known, this can be estimated as the number of visited users who tagged i with t.

The choice between the two possible execution branches can rely on heuristics which estimate their utility w.r.t approaching the final result. Two such heuristics are explained in [Maniu and Cautis 2013b; Schenkel et al. 2008], guiding this choice either by estimating the maximum potential score of each branch, or by choosing the branch that is the most likely to refine the score of the item outside the current top-k which has the highest estimated score (a choice that is likely to advance the run of the algorithm closer to termination).

4.3. Adaptations for incremental computation

We now describe the extension to perform the as-you-type computation *incrementally*, as follows:

(1) when a new keyword is initiated (i.e., t_r is one character), we take the following steps in order:
  (a) purge all D_t buffers for $t \in C$, except for $D_{t_{r-1}}$ (t_{r-1} is no longer a potential prefix, but a complete term),
  (b) reinitialize C to the empty set,
  (c) purge all CIL_t buffers for $t \notin \{t_1, \dots, t_{r-1}\}$,
  (d) reinitialize the network exploration (the queue H) to start from the seeker, in order to visit again p-spaces looking for triples for the new prefix, t_r. (This amounts to the following

changes in Algorithm 1: among its initialization steps (1-12), steps (4-8) are removed, and steps for points (a) and (c) above are added.)

(2) when the current t_r is augmented with one additional character (so t_r is at least two characters long), we take the following steps in order:
   (a) purge D_t buffers for $t \in C$ s.t. t is not a t_r completion
   (b) remove from C all ts which aren't completions for t_r,
   (c) purge all CIL_t buffers for $t \notin \{t_1, \ldots t_{r-1}\} \cup C$,
   (d) resume the network exploration.
   (This amounts to the following changes in Algorithm 1: among its initialization steps (1-12), steps (4-8) and (10-12) are removed, and steps for points (a), (b), and (c) above are added.)

Note that, in the latter case, we can efficiently do the filtering operations by relying on a simple trie structure for the C set and use it as the index for for directly accessing the other data structures (D-lists and CIL-lists).

4.4. Complexity analysis

Recall that $Tagged(v, i, t)$ denotes the set of unique triples and consider a query $Q = (t_1, \ldots, t_r)$. Let f denote the average fan-out in the CT-IL index, d_r the average depth of the trie subtree rooted at the node corresponding to t_r (models the size of t_r completions), and p the average p-space size.

CT-IL is a space-efficient trie structure for sorted access, as a node can represent a sequence of characters. Thus, the memory space to store the trie is reduced compared to the trie index of [Li et al. 2012]. Given an item i and a tag t, there corresponds a unique entry $(i, tf(i, t))$ in $IL(t)$. In total, there are as many entries in inverted lists as ***unique*** pairs (i, t), leading to a total space for the inverted lists at the leaves of CT-IL of $\mathcal{O}(|\{unique\ (i, t)\}|)$. The number of inverted lists corresponds to the number of distinct tags in the vocabulary, $|\mathcal{T}|$. For example, in the case of our Yelp dataset, there are $177, 286$ such lists and a simple computation reveals that in average each would have approximately 70 entries.

For each user u, we store a p-space index containing all pairs (i, t) of u. Thus, each triple is indexed in p-spaces exactly once. The shortest-path computations for exploring the social graph by order of proximity is straightforwardly $\mathcal{O}(|E| + |\mathcal{U}| \log |\mathcal{U}|)$.

In the run of TOPKS-ASYT, we visit one user at a time in the social graph and, in the worst-case, we may visit the entire network, unless an event like the termination condition, a keystroke, or (most likely) the time limit occurs.[4] While entries in inverted lists are read at most once (see Algorithm 3), the situation is different for p-spaces, as they may need to be explored once for each new word that is added to the query (see step (1)(d) from the previous section), leading to $O(r)$ network explorations.

A one-character search (i.e., expansion of t_r) initially costs $\mathcal{O}(f)$ in CT-IL, and is followed by a sequence of variable length of sorted accesses in the trie and in the social graph; their actual number depends on the value of α and on the overlap between p-spaces and CT-IL. Individually, the former accesses have a direct cost of $\mathcal{O}(d_r \log d_r)$. However, compared with the compact-trie of [Hsu and Ottaviano 2013], this direct cost we incur is roughly double (albeit reduced), since our leaves are not necessarily single strings, but lists thereof, and thus a sorted access in a priority queue most often will translate in a score update instead of a normal pop operation.

Just like the NRA algorithm of [Fagin et al. 2001], whose complexity is quadratic in the size of its buffers, the bookkeeping steps are more expensive complexity-wise because score intervals are maintained throughout the computation, so we cannot have bounded buffers for our candidates. Whenever the p-space of some user u is visited (Algorithm 2), for a given completed tag $t \in Q$ used by u, the cost of the updates to be done on the buffer D_t is $\mathcal{O}(p|D_t|)$; an additional cost of $\mathcal{O}(p \sum_t |D_t|)$, for t denoting t_r completions, corresponds to the tag t_r still in the typing. Regarding accesses to CT-IL by Algorithm 3, the cost is of the order $\mathcal{O}(p(|D_t|^2 + |CIL_t||D_t|))$ for the completed tags t, and $\mathcal{O}(p \sum_t (|D_t|^2 + |CIL_t||D_t|))$ for the completions t of t_r. Overall, in the

[4]It is highly likely in practice that typing latency precludes most often a computation until termination conditions are reached.

Table I. Statistics on the datasets we used in our experiments.

	Twitter	Amazon	Tumblr	Yelp
Number of unique users	$458,117$	$130,098$	$612,425$	$29,293$
Number of unique items	$1.6M$	$252,891$	$1.4M$	$18,149$
Number of unique tags	$550,157$	$91,352$	$2.3M$	$177,286$
Number of triples	$13.9M$	$24.7M$	$11.3M$	$30.3M$
Average number of tags per item	8.4	53.8	7.9	685.7
Average tag length	13.1	6.9	13.0	6.5

most important case for our study – exclusively social, i.e., $\alpha = 0$, for one prefix query, i.e., $r = 1$ – the worst-case time complexity of our algorithm is $\mathcal{O}(|E| + |\mathcal{U}| \log |\mathcal{U}| + d_r|\mathcal{U}|p \sum_t(|D_t|^2 + |CIL_t||D_t|)$, for the completions t of t_r.

Compared to the non-incremental version, the algorithm avoids to restart the graph exploration from the seeker s and simply continues from the currently visited node. As described in Section 4.3, the pruning of all unnecessary data structures D_t, CIL_t, and C – for t denoting here the previous completions that do not match newly typed letter, can be done efficiently in $\mathcal{O}(f)$ by using a trie for the C-set, which can act as the vocabulary index leading to the D_t and CIL_t buffers.

4.5. Supernodes

When visiting a user node, we need to explore its p-space – its tag contributions – by routine PROCESS_P_SPACE (Algorithm 2). This can be costly overall if p-spaces are saved on disk, since many p-spaces may be loaded in main memory. In the case of time-limited queries, when a budget is imposed (e.g., in terms of random disk accesses) and results must be returned before budget expiration, loading p-spaces from disk becomes therefore a core issue. In this section, we discuss a way to make p-space exploration go deeper in the graph, under access budget constraints.

Most sequences of users visited by TOPKS-ASYT are unique to each seeker. Thus, unless each possible sequence was materialised and cached on disk, p-spaces must be loaded one at a time. To tackle this issue, we propose to cluster users into *supernodes* and apply TOPKS-ASYT on the graph reduced to supernodes. Instead of loading one p-space at a time, several p-spaces included in the same supernode can be loaded jointly, with the tradeoff of some limited "off-track" exploration.

To build N supernodes, we first select N random users in the graph. Each user will correspond to the centroid of a supernode. Every remaining vertex u is then assigned to the supernode whose centroid is the closest to u. This method has the advantage of producing supernodes of relatively balanced sizes, which is exactly the purpose of clustering users into supernodes. Obviously, if the cluster sizes were unbalanced, that would make TOPKS-ASYT perform considerably worse when having to load many small supernodes. (Indeed, in preliminary experiments, state-of-the-art community detection assigned most users to few supernodes, letting most other supernode cardinalities far under the average number of users per cluster; this is why we followed a different user grouping.)

5. EXPERIMENTS

We evaluate in this section the effectiveness, scalability, and efficiency of TOPKS-ASYT. We used a Java implementation of our algorithms, on a low-end Intel Core i7 Linux machine with 16GB of RAM. We performed our experiments in an all-in-memory setting, for datasets of medium size (10-30 millions of tagging triples). We describe first the applications and datasets we used for evaluation.

5.1. Datasets

We used several popular social media platforms, namely Twitter, Tumblr, Yelp, and Amazon, from which we built sets of (user, item, tag) triples. Table I reports some statistics about each dataset.

Twitter: tagging triples. We used a collection of tweets extracted during Aug. 2012. As described in Section 3, we see each tweet and its re-tweet instances as one item, and the authors of the tweets/re-tweets as its taggers. We include both the text and the hashtags as tags.

Amazon movies: tagging triples. We used a publicly available SNAP dataset of around 35 million movie reviews, spanning a period of 18 years up to March 2013. In this social media scenario, in order to build the user-item-tag triples, we simply considered the movie as the item, the author of the review as the tagger, and the keywords appearing in the review as the tags.

Tumblr: tagging triples and social links. We extracted a collection of Tumblr posts from Oct.-Nov. 2014, following the same interpretation on posts, taggers, and tags as in Twitter. Among the 6 different types of posts within Tumblr, we selected only the *default* type, which can contain text plus images. Moreover, in the case of Tumblr, we were able to access the follower-followee network and thus we extracted the induced follower-followee network for the selected taggers.

Yelp: tagging triples and social links. Lastly, we considered a publicly available Yelp dataset, containing reviews for businesses and the induced follower-followee network[5]. In this case, in order to build the triples, we considered the business (e.g., restaurant) as the item, the author of the review as the tagger, and the keywords appearing in the review as the tags.

For Twitter and Tumblr, to enrich the set of keywords associated to an item, we also expand each tag by the at most 5 most common keywords associated with it by a given user, i.e., by the tag-keyword co-occurrence. Finally, from the resulting sets of triples, we removed those corresponding to (i) items that were not tagged by at least two users, or (ii) users who did not tag at least two items.

To complete the data setting for our algorithm, we then constructed the user-to-user weighted networks that are exploited in the social-aware search. For this, we first used the underlying social network (when available). Specifically, for each user pair in Tumblr or Yelp, we computed the Dice coefficient corresponding to the common neighbors in the social network. To also study situations when such a network may not be available (as for Twitter and Amazon), exploiting a thematic proximity instead of a social one, we built two other kinds of user similarity networks, based on the Dice coefficient over either (i) the item-tag pairs of the two users, or (ii) the tags of the two users. We considered the filtering of “noise” links, weighted below a given cut-off threshold. Among the resulting ten networks, the Amazon tag similarity one was discarded due to poor connectivity coupled with high density and thus a less discriminative nature; we therefore report next on nine different network configurations.

5.2. Experimental results: effectiveness

We present in this section the results we obtained in our experiments for effectiveness, or “prediction power”, with the purpose of validating the underlying as-you-type query model and the feasibility of our approach. In this framework, for all the data configurations we considered for effectiveness purposes, we imposed wall-clock time thresholds of $50ms$ per keystroke, which we see as appropriate for an interactive search experience.

To measure effectiveness, we followed an assumption used in recent literature, e.g. in [Pennacchiotti et al. 2012; Maniu and Cautis 2013b], namely that a user is likely to find his items – belonging to him or re-published by him – more interesting than random items from other users. For testing effectiveness, we randomly select triples (u, i, t) from each dataset. For each selected triple, we consider u as the seeker and t as the keyword issued by this user. The aim is to “get back” item i through search. The as-you-type scenario is simulated by considering that the user issues t one letter at a time. Note that an item may be retrieved back only if at least one user connected to the seeker tagged it. We picked randomly 800 such triples (we denote this selection as the set D), for tags having at least three letters. For each individual measurement, we gave as input a triple *(user, item, tag)* to be tested (after removing it from the dataset), and then we observed the ranking of *item* when *user* issues a query that is a prefix of *tag*.

Note that we tested effectiveness using single-word search for Twitter, Tumblr, and Amazon. On the contrary, for Yelp, due to its distinct features of having many triples per user, we did two-word search: given a query $q = (w_1, w_2)$, we first filtered items tagged by w_1, we then processed the remaining triples with query w_2 in the same manner as we did for Twitter and Tumblr.

[5]http://www.yelp.com/dataset_challenge

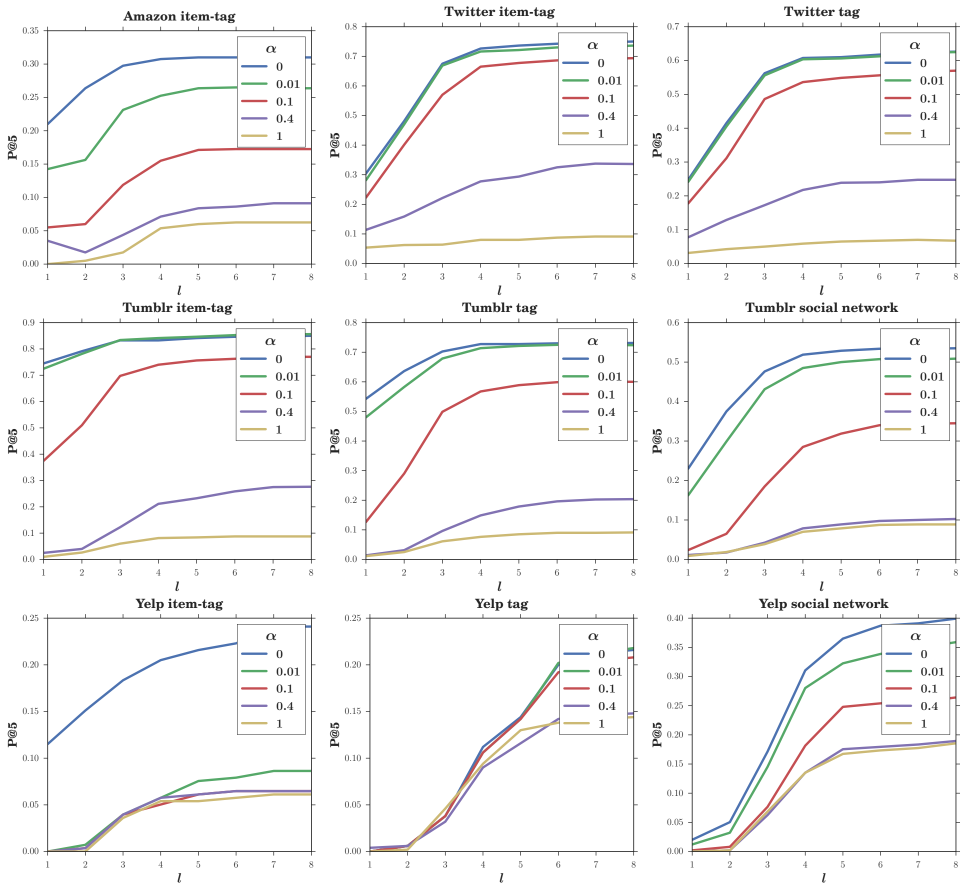


Fig. 5. Impact of α on precision.

We define the precision $P@k$ for our selected set D as

$$P@k = \frac{\#\{triple \mid ranking < k, triple \in D\}}{\#D}$$

Since $P@k$ can be seen as a function of the main parameters of our system, one goal was to understand how it is influenced by them. We describe now the different parameters we took into account.

— l, length of the prefix in the query (number of characters).
— θ, the threshold used to filter similarity links keeping only those having a score above.
— α, the social bias ($\alpha = 0$ for exclusively social score, $\alpha = 1$ for exclusively textual score).
— $\eta_i(u)$, the number of items tagged by user u, a user activeness indicator (for simplicity, hereafter referred to as η_i).
— $\eta_u(i)$, number of users who tagged item i, an item popularity indicator (η_u).

We present next the results we obtained for this experiment. (For space reasons, we only report here on $P@5$, but we performed test with $P@1$ and $P@20$ as well, which showcase similar evolution and improvement ratios, in the case of the latter, most often reaching precision levels of around 0.8-0.9.) When parameters are not variables of a figure, they take the following default values: $\alpha = 0$ (fully social bias), θ is assigned the lowest value of the tested dataset, η_i and η_u are associated to active users and popular items ($\eta_i \geq 3$ and $\eta_u \geq 10$).

Impact of α. As shown in Figure 5, α can have a major impact on precision. With a fully social bias ($\alpha = 0$), we obtained the best results for the four datasets and all the available similarity

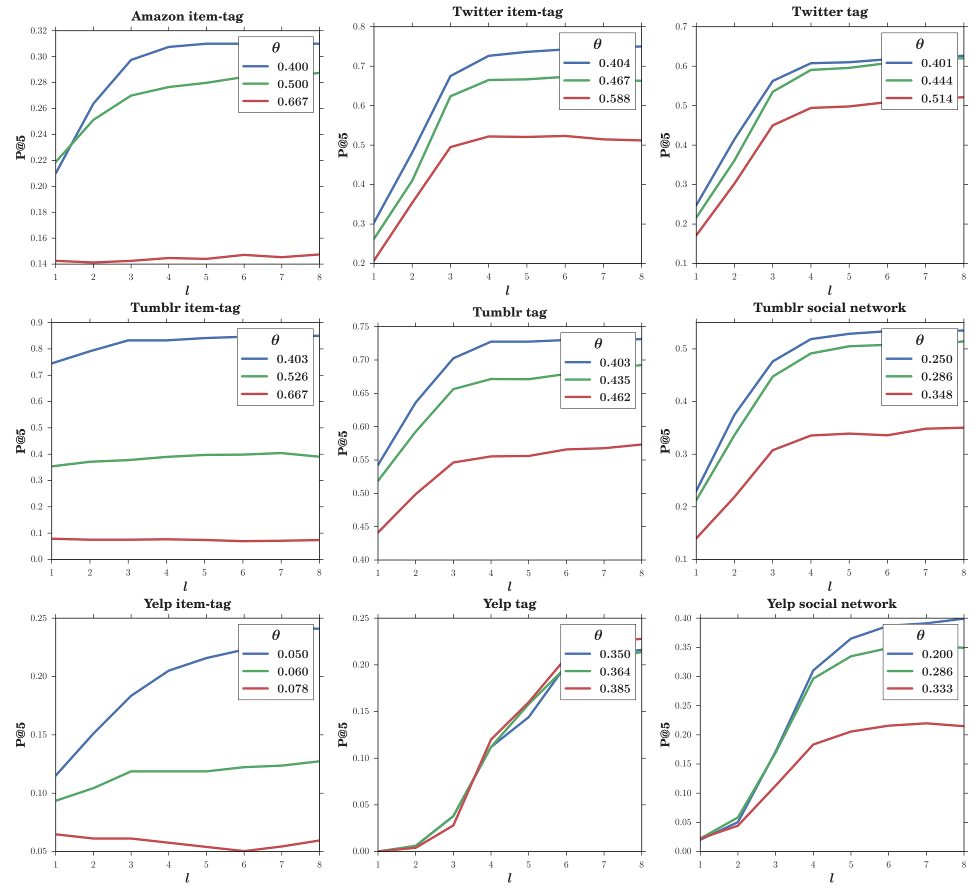


Fig. 6. Impact of θ on precision.

networks. Moreover, typing new characters to complete the prefix increases the precision. However, the evolution for $\alpha = 0$ can be quite slow, with the Tumblr or Yelp item-tag similarity network for witness. In this case, one likely reason is that these networks are quite rich in information, and the neighbors of the seeker are very likely to have the searched item, with the right tag, due to the way this network was built. This can also explain why the precision for the item-tag networks is higher in the case of Tumblr than those for tag and social similarity networks. The precision for the social similarity network is the lowest for Tumblr, while in the case of the Yelp dataset the best results are obtained using the social network. Recall that the tag and item-tag networks were built based on the same content we were testing on, whereas the social similarity network only uses the links between users to infer distances between them. Yelp and Amazon exhibit lower precision levels overall, unsurprisingly, since they are denser datasets (more triples per user).

Interestingly and supporting our thesis for social bias, we obtain good precisions levels with such networks of similarity in social links (the highest in the case of Yelp). For example, in the case of Tumblr, we can reach $P@5$ of around 0.82 for the item-tag similarity network, 0.7 for the tag one, and still 0.5 for the social one. This indicates that we can indeed find relevant information using a content-agnostic network using TOPKS-ASYT. Importantly, it also indicates that we can always search with the same social similarity network, even when the content evolves rather rapidly, with the same precision guarantees.

Finally, we can see that the evolution curve for small α values, as new characters are added, varies depending on the similarity network. In Tumblr for example, the precision for low α values does not increase much using the item-tag similarity network. The items were found very close to the seeker

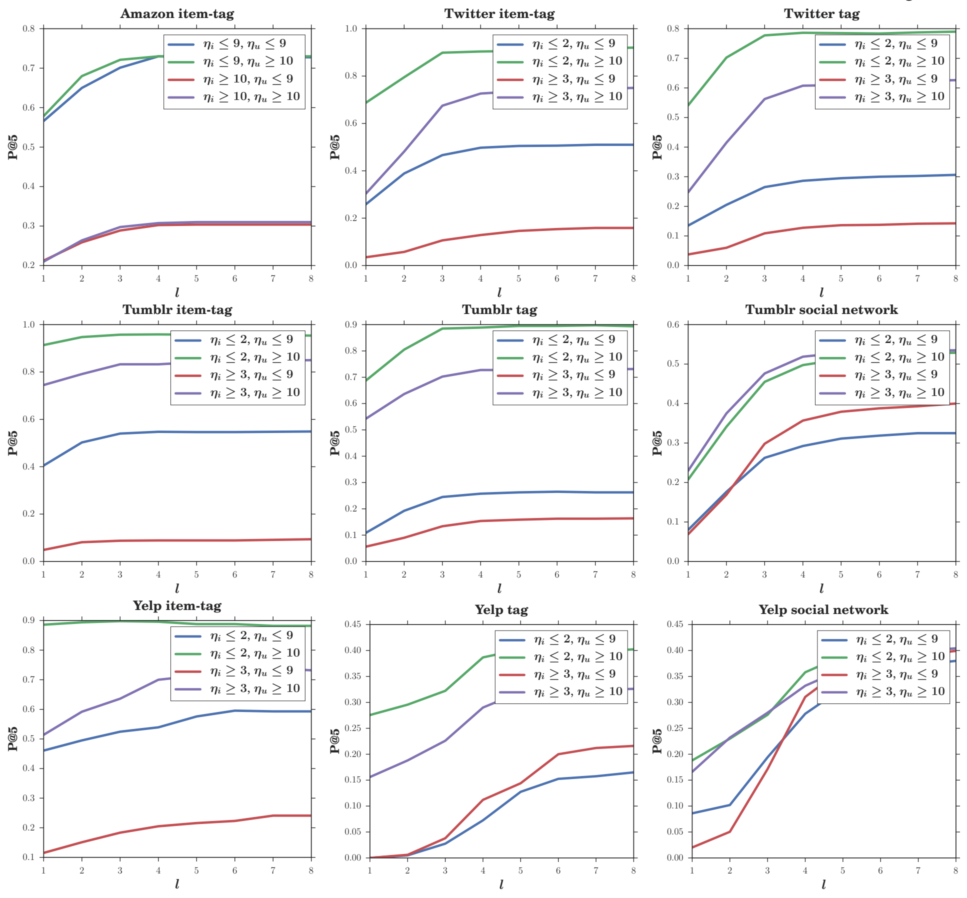


Fig. 7. Precision for various types of users and items.

and a few characters already give the final score in most cases. Very likely, the average number of items per user is too low to make the length of the prefix have an impact (most probably, users close to the seeker would not have several items tagged with the same prefix, even if this prefix is short). On the contrary, with the social similarity network, items with a tag matching the prefix are more likely to be diverse around the seeker. The distribution of the searched item in the network should thus be less concentrated around the seeker. Therefore, the number of result candidates with a high score for short prefixes is larger, and increasing l has more impact on the precision. Whereas an item-tag network tends to do so by definition, this can be seen as a clear consequence of the social bias that motivated our work.

Impact of θ. In Figure 6, we can observe the impact of θ on the quality of results. We mention that the two highest θ values lead to 33% and 66% cuts on the total number of edges obtained with the lowest θ value. Unsurprisingly, removing connections between users decreases the precision. When using the similarity network filtered by the lowest θ value, the seeker is almost always connected to the network's largest connected component, and we can visit many users to retrieve back the targeted item. With higher θ values, the connectivity for certain seekers we tested with is broken, making some of the tested items unreachable.

Impact of popularity / activeness. We show in Figure 7 the effects of item popularity and user activity. For all similarity networks, the precision is better for popular items (high η_u). This is to be expected, as a popular item is more likely to be found when visiting the graph, as it is expected that it will score high since it has many taggers. Along with item popularity, we can observe that user activeness has a different effect in both content-based and the social similarity networks. Active users

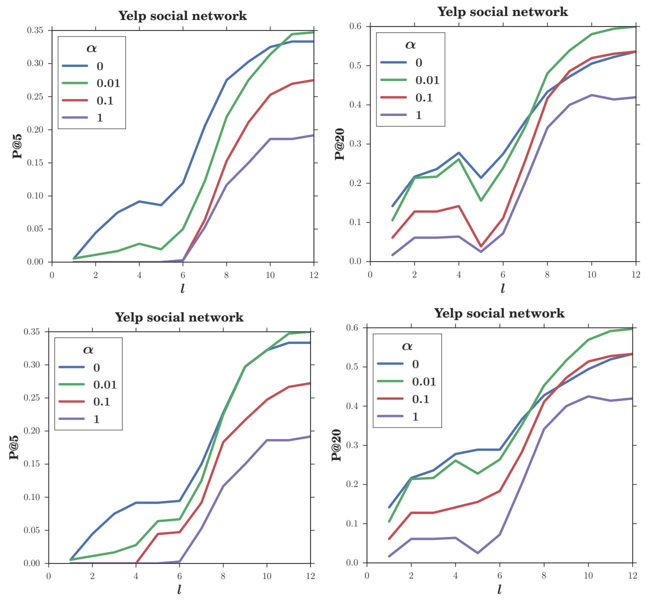


Fig. 8. Precision for multiple-word queries without (top row) and with (bottom row) correction.

yield a better precision score when similarity comes from social links, whereas it is the opposite with content-based similarity networks. Reasonably, retrieving back an item for a non-active seeker in a content-based network is easier since his similarity with neighbors is stronger (Dice coefficient computed on less content).

5.3. Experimental results: effectiveness with multiple words

We describe next an additional experimental evaluation for effectiveness, focusing on the case of multiple word queries, in the densest dataset (Yelp). When dealing with multi-word queries, the score of the prefix can have a highly disproportionate weight compared to other terms in the query.

As we take the maximal score over all the possible completions (Eq.(6), (7)), the score for short prefixes is likely to be very high and render irrelevant the preceding terms. For example, if the query is composed of two terms t_1 and t_2, since t_2 is interpreted as a prefix, its length can influence the expectation of its score (for any value of α). Say t_2 is a prefix of length 2, it is very likely to be the root of many possible completions; thus the expected value of the maximal score over all completions will likely be much larger than the score of t_1. Furthermore, note that short prefixes bring little information about a seeker's intent.

Figure 8 top row shows the values for precision for multi-word queries in Yelp, *without correcting the score of the last term*. The first four letters ($l = 1, \ldots, 4$) correspond to the last letters of the first word. The following characters ($l \geq 5$) correspond to the next word. As expected, due the effect described above, we can see drops in the precision when starting a new word ($l = 5$) for any value of α. The precision loss is particularly observable when measuring $P@20$. This motivated the following model adjustment.

To make the score of a small prefix comparable to those of the preceding terms, we propose to re-scale it by a data-dependent constant. Specifically, for each prefix length ($l \geq 1$), we compute a normalizer value that maximized the precision through cross-validation. For example, we computed the parameter N_1 (i.e., the normalizer of prefixes of length 1) to optimize the precision of queries of the form $q = (t_1, p)$, where p is a prefix of length 1. Proceeding similarly for other prefix lengths, in Yelp we obtained constants $N_1 = 10^3, N_2 = 200, N_3 = 50, N_4 = 20, N_5 = 8$ and $N_6 = 2$.

In Figure 8 bottom row, with the normalized scores, we can observe that the drop of precision seen in Figure 8 has almost disappeared (see the case of $P@20$, where we had significant drops when starting a new term, but the correction with N_1 now preserves precision).

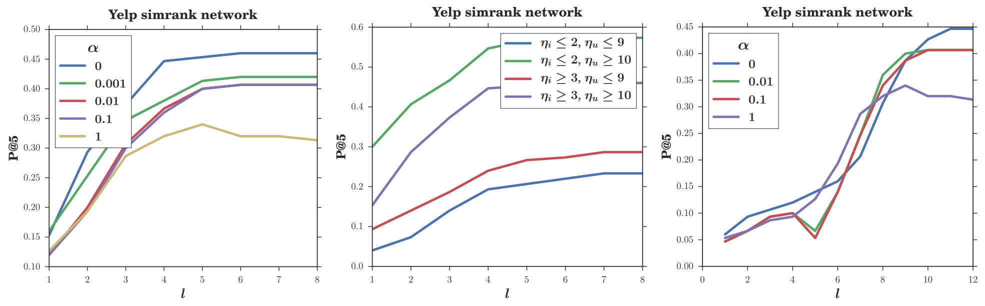


Fig. 9. Impact of α (left), type of users / items (middle), multiple words with correction (right) with SimRank proximity (for comparison with Figures 5, 7, and 8 bottom row respectively).

5.4. Experimental results: effectiveness with SimRank proximity scores

We conducted similar experiments for effectiveness using, instead of the neighborhood-based Dice proximity extended to shortest paths, the well-known path-based proximity model SimRank. For space reasons and to avoid repetition, we highlight the results over the densest dataset (Yelp), for comparison with all the initial plots for effectiveness. This allowed us to observe how the chosen similarity (local/single-path or global) impacts results. The SimRank model, introduced by [Jeh and Widom 2002], gives a recursive definition of the similarity between users u and v as follows:

$$\sigma^{SimRank}(u,v) = \frac{c}{|N_u||N_v|} \sum_{u' \in N_u, v' \in N_v} \sigma^{SimRank}(u',v') \qquad (8)$$

for some decay factor $c \in [0,1]$. (A similar definition can be given for directed graphs.) The thesis is that “two objects are considered to be similar if they are referenced by similar objects”. Since this definition is recursive, the SimRank score between two users depends on the whole graph.

In Figure 9, we used SimRank similarity computed on the social network instead of the Dice's coefficient used before. On the left figure, we display the impact of α on precision and observe the best results using a fully social bias. Interestingly, we have a slight improvements using SimRank similarity as it reaches a precision P@5 of 0.45 after typing 5 letters, which is to compare to the value 0.35 observed with Dice's coefficient. This supports the use of path-based similarity measures that encompass the general relationships between nodes.

On the right figure, we observe the impact of user activeness and item popularity on precision. Once again, results are better for popular items. Note that SimRank has no cut-off threshold preventing us from experimenting the impact of θ.

5.5. Experimental results: efficiency and scalability

We now turn our attention to the efficiency and scalability aspects of our solutions. In Figure 10 top row, we display the evolution of NDCG@20 vs. time, for the densest dataset (Yelp), for different α values (where α is normalized to have similar social and textual scores in average). The NDCG is computed w.r.t. the exact top-k that would be obtained running the algorithm on the entire graph. This measure is an important indicator for the feasibility of social-aware as-you-type search, illustrating the accuracy levels reached under “typing latency”, even when the termination conditions are not met. In this plot, we fixed the prefix length size to $l = 4$. The left plot is when a user searches with a random tag (not necessarily used by her previously), while the right plot follows the same selection methodology as in Section 5.2. Importantly, with α corresponding to exclusively social or textual relevance, we reach the exact top-k faster than when combining these two contributions ($\alpha = 0.5$). Note also that this trend holds even when the user searches with random tags.

In Figure 10 bottom row, we show the evolution of NDCG@20 vs. time in Yelp, for different prefix lengths (the left plot is for random tags). Results shows that with lower l values we need more time to identify the right top-k. The reason is that shorter prefixes can have many potential (matching) items, hence the item discrimination process evolves more slowly. For these prefix lengths, we only mention here that we also analyzed the evolution of NDCG@20 when visiting a *fixed number*

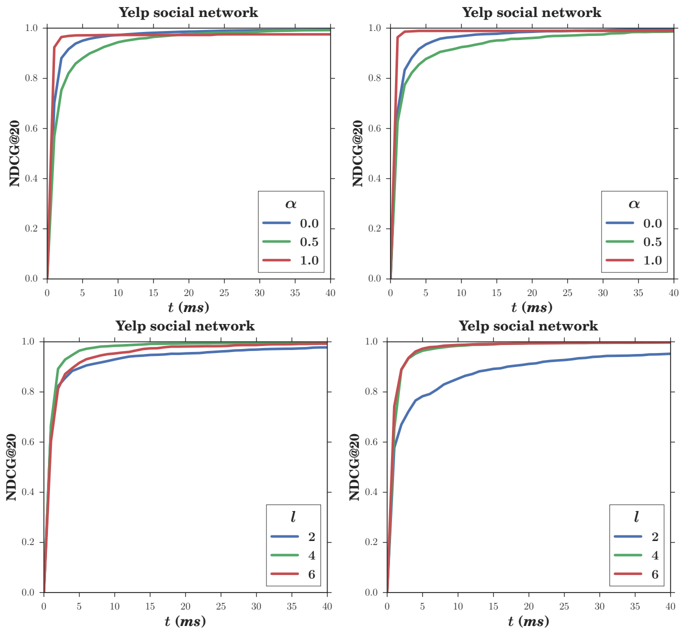


Fig. 10. Impact of α (top) or l (bottom) on NDCG vs time for random search (left) or personal search (right).

number of users, observing similar behavior. As expected, the more users we visit the higher NDCG we reach and for longer prefixes it is necessary to visit more users. E.g., when $l = 6$, after visiting 500 users, we reach an NDCG of 0.8 while for $l = 2$ the NDCG after 500 visits is 0.9.

Finally, in the experiment illustrated in Figure 12 we observed the time to reach the exact top-k for different dataset sizes. For that, we partitioned the Yelp triples sorted by time into five consecutive (20%) chunks. For each dataset we perform searches using prefixes of $l = 2, 3, 4, 5$. While the time to reach the exact top-k increases with bigger datasets and shorter prefixes, the algorithm scales adequately when l is more than 2. For instance, for $l = 3$, the time to reach the result over the complete dataset is just twice the time when considering only 20% of this dataset.

5.6. Experimental results: incremental versus non-incremental TOPKS-ASYT

We now analyze the impact of the incremental computation. In Figure 13, we display the time to reach the exact top-k for both TOPKS-ASYT and its incremental counterpart. For that, we compare the two algorithms on sequences of consecutive prefixes, e.g. sou, $sour$, $sourc$, and $source$. Let p_t and p_{t+1} be two consecutive queries differing by a single character. Whereas TOPKS-ASYT starts a new query for each new letter, the incremental version calculates the answer for p_{t+1} relying on computations for p_t. Obviously, the time to reach the exact top-k for the first prefix p_1 is the same (the same algorithm is run). For $l = 4$, the time to reach the result is already slightly smaller for the incremental version of the algorithm. We emphasize that the first part of the incremental algorithm, which consists in filtering the previous candidate list explains the small improvement. For longer prefixes ($l = 5, 6$), the candidate list is shorter and the incremental algorithm takes full advantage of previous computations (speed increase from $\times 2$ for $l = 5$ up to $\times 4$ for $l = 6$).

5.7. Experimental results: TOPKS-ASYT versus state-of-the-art baseline methods

We compare TOPKS-ASYT with three different baselines methods. The first two methods respectively build on the state-of-the-art social top-k search TOPKS algorithm from [Maniu and Cautis 2013b] and the type-ahead textual search algorithm NRA(HEAP) of [Li et al. 2012][6]. The third method relies on the online Yelp Search API with query-autocompletion.

[6]This method was implemented and made available by the authors, as part of an instant-search engine called SRCH2; its source code is available at https://github.com/SRCH2/srch2-ngn.

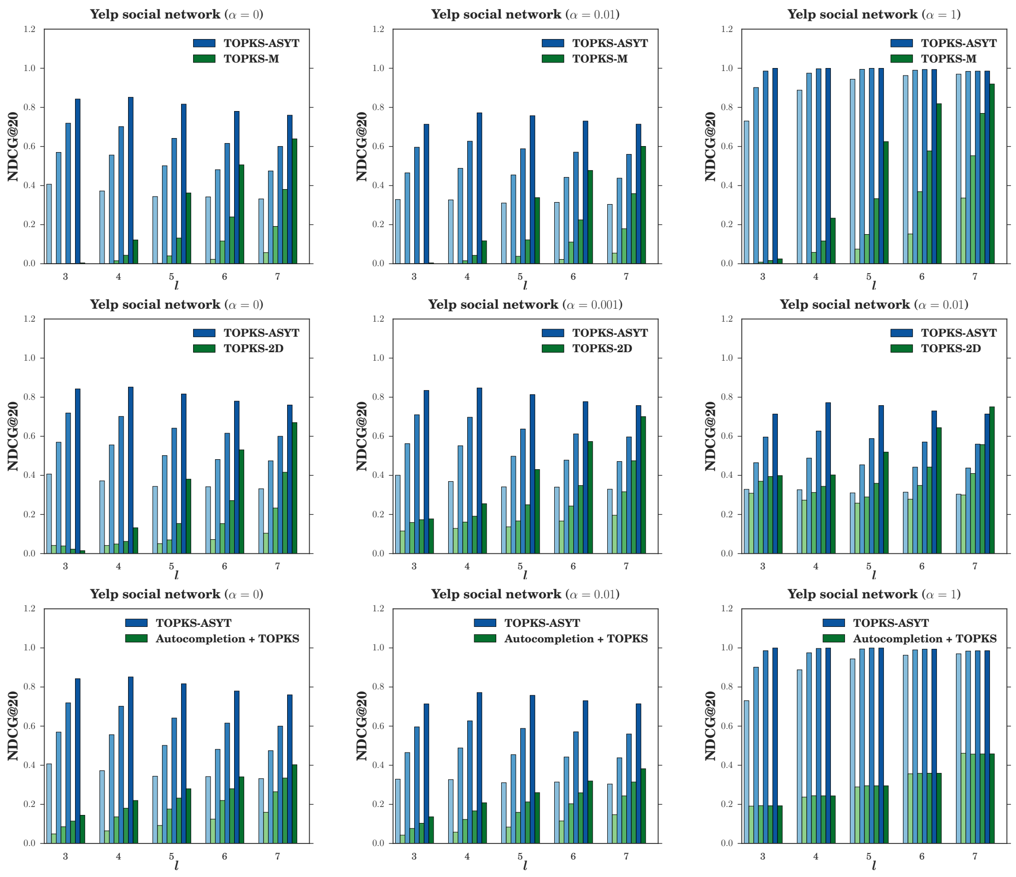


Fig. 11. TOPKS-ASYT vs. TOPKS-M (top), TOPKS-2D (middle) or Autocompletion + TOPKS (bottom) baselines with fixed budget.

TOPKS-M*: social top-k baseline.* We first compare TOPKS-ASYT to TOPKS-M (for TOPKS-Merge), a baseline method that follows a natural idea relying on the social top-k state-of-the-art (such as algorithm TOPKS from [Maniu and Cautis 2013b]), but does not benefit from CT-IL (i.e., does not benefit from prefix-based retrieval). The approach of TOPKS-M works in two stages, as follows: first, we load the inverted lists of all the possible completions of the final term given in the query and merge them in a unique list. As a result, this step may be very costly for short prefixes. Once the first step is completed, we can directly apply algorithm TOPKS, using prefixes as complete words with their own inverted list.

In Figure 11 top row, we show the NDCG@20 of TOPKS-ASYT and TOPKS for various budgets (50, 100, 200, 400 – each value corresponds to a color intensity, from lighter to darker) and various α. A budget B corresponds to the maximal number of *significant disk accesses* we allow the algorithm to do to answer a query. In our interpretation, a significant disk access can be either a p-space exploration (visit of the next user in the algorithm) or the loading of an inverted list. Our experiments show that the second is more costly ($\times 12$ in average), thus, we count a budget consumption of 12 for a disk access corresponding to an entire inverted list. Similarly to Section 5.5, the NDCG is computed w.r.t. the exact top-k that would be obtained by running the algorithm on the entire similarity graph without budget restrictions.

For α ranging from 0 to 1, we observe a similar behavior, when comparing TOPKS-ASYT to the baseline method. Results show that the NDCG of TOPKS-M is much smaller than the one of TOPKS-ASYT, even for relatively important budgets (e.g., $B = 400$). The cost of merging inverted lists before applying TOPKS prevents the algorithm from providing high-quality answers fast. For example, for short prefixes ($l = 3, 4$), too many completions are possible and thus the baseline loads too many inverted lists compared to the budget. Even for a budget $B = 400$, TOPKS cannot catch up with the precision of TOPKS-ASYT for a prefix length $l = 3$.

TOPKS-2D*: textual search as-you-type baseline.* We now compare TOPKS-ASYT to a *dimension-at-a-time* approach, denoted TOPKS-2D (for TOPKS-2-Dimensions), which processes social and textual contributions separately. First, we retrieve documents matching the query on the textual dimension using the NRA(HEAP) type-ahead baseline from [Li et al. 2012]. That is, we read inverted lists and build the candidate list, ignoring the social contribution (we do not use the social graph and no p-space is explored). In a second stage, we then explore the similarity graph to obtain the social contribution in the final score.

In Figure 11 middle row, for the same budget values and color code as before, we show the NDCG@20 of TOPKS-ASYT and TOPKS-2D, for various values of α ($\alpha = 1$ is not considered, as it is a case where TOPKS-ASYT and the baseline are virtually the same). We can see that small values of α highly favor TOPKS-ASYT: NRA(HEAP) spends useless budget on inverted lists, since it runs without knowledge on the social scores. On the contrary, TOPKS-ASYT benefits from simultaneous social and textual score computations to avoid using unnecessary inverted lists. When α increases, the textual contribution becomes more significant and the baseline method becomes more competitive, especially for longer prefixes that do not have many possible completions.

AUTOCOMPLETION+TOPKS *baseline.* We complete our performance comparison with AUTOCOMPLETE+TOPKS, a baseline method that relies on the Yelp Search API for query-autocompletion[7]. This baseline method proceeds as follows: we obtain a set of queries that are predicted by Yelp to complete the current query (prefix) the seeker is typing, without using any social information (the service is not "personalized"). We then use these queries to get the set of top-k results over our data, by simply running for them the aforementioned state-of-the-art network-aware top-k algorithm TOPKS from [Maniu and Cautis 2013b]. We give TOPKS-ASYT and the baseline the same budgets as in previous experiments and, to avoid any potential evaluation bias in our favor, any costs from the Yelp API auto-completion step are ignored.

In Figure 11 bottom row, we show the NDCG@20 of TOPKS-ASYT and AUTOCOMPLETE+TOPKS using the same display convention as before. We can see that for all values of α, the NDCG of AUTOCOMPLETION+TOPKS is significantly smaller than the one of TOPKS-ASYT. As the API does not use social information to construct autocompletions, the final top-k is likely affected by the general query trend and this should explain the NDCG for low values of α. Interestingly, we can however observe a similar behavior even for high values of α (textual score).

5.8. Experimental results: supernodes

In Figure 14, we show the impact of the supernode materialization feature, for supernodes of average size $d = 6$ and $d = 30$. For three different budgets ($B = 10, 30, 50$), we run TOPKS-ASYT with the original similarity network and the supernode-reduced graph. Similarly to the previous section, the budget corresponds to the number of significant disk accesses we allow our algorithm to do until it outputs a top-k result. For budget $B = 50$, supernodes do not increase the NDCG, in particular for short prefixes. Small prefixes have many completions and thus are very common. This means that most of the NDCG contribution is obtained with few visited nodes. When the budget given to TOPKS-ASYT is smaller, supernodes improve the ranking quality. For instance, with budget $B = 10$, very few nodes can be visited by TOPKS-ASYT and the supernodes become a key feature. With supernodes of 6 users (resp. 30), the algorithm aggregates p-spaces of up to 60 people (resp. 300), whereas it would visit at most 10 neighbors using the initial similarity network.

Main-memory vs. secondary memory considerations. We emphasize here that we performed our experiments in an all-in-memory setting, for datasets of medium size (tens of millions of tagging

[7]https://github.com/Yelp/yelp-api-v3/blob/master/docs/api-references/autocomplete.md

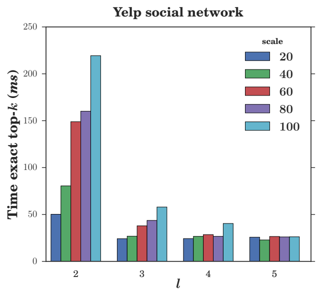


Fig. 12. Time to exact top-k for different dataset sizes.

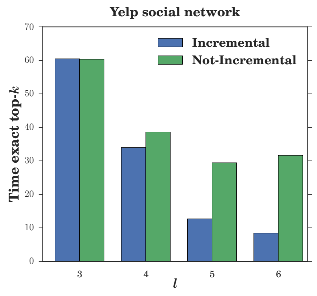


Fig. 13. Incremental vs non-incremental version.

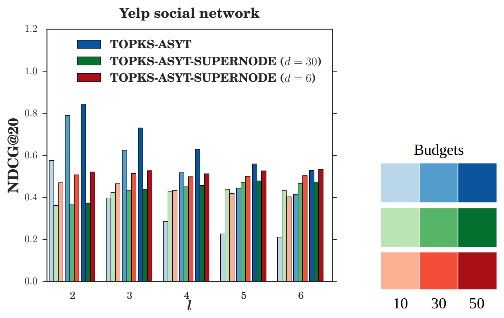


Fig. 14. Impact of supernode sizes on NDCG for several prefix lengths.

triples), in which the advantages of our approach may not be entirely observed. In practice, in real, large-scale applications such as Tumblr, one can no longer assume a direct and inexpensive access to p-spaces and inverted lists, even though some data dimensions such as the user network and the top levels of CT-IL – e.g., the trie layer and possibly prefixes of the inverted lists – could still reside in main memory. In practice, with each visited user, the search might require a random access for her personal space, hence the interest for the sequential, user-at-a-time approach. Even when p-spaces may reside on disk, a previous experiment shows that by retrieving a small number of them, less than 100, we can reach good precision levels; depending on disk latency, serving results in, for example, under $100ms$ seems within reach. One way to further alleviate such costs may be to cluster users having similar proximity vectors, and choose the layout of p-spaces on disk based on such clusters; this is an approach we intend to evaluate in the future, at larger scale.

6. CONCLUSION

We study in this paper as-you-type query search in social media applications. In particular our aim was to retrieve the top-k ranked results, under a network-aware query model by which information produced by users who are closer to the seeker can be given more weight. We formalize this problem and we describe the TOPKS-ASYT algorithm to solve it. Our solutions is based on a novel trie data structure, CT-IL, allowing ranked access over inverted lists. In several application scenarios, we perform extensive experiments for efficiency, effectiveness, and scalability, validating our techniques and the underlying query model. As a measure of efficiency, since as-accurate-as-possible answers must be provided while the query is being typed, we investigate how precision evolves with time and, in particular, under what circumstances acceptable precision levels are met within reasonable as-you-type latency (e.g., less than $100ms$). Also, as a measure of effectiveness, we analyze thoroughly the “prediction power” of the results produced by TOPKS-ASYT. We intend to compare these results in the future with the ones from an online evaluation, which could illustrate directly how the social bias in the retrieval process can affect CTR for search.

We see many promising directions for improving the TOPKS-ASYT algorithm. First, for optimizing query execution over the CT-IL index structure, we intend to study how CT-IL can be enriched with certain pre-computed unions of inverted lists (materialized virtual lists). Assuming a fixed memory budget, this would be done for chosen nodes (prefixes) in the trie, in order to speed-up the sorted access time, leading to a memory-time tradeoff. While similar in spirit to the pre-computation of virtual lists of [Li et al. 2012], a major difference for our setting is that we can rely on a materialization strategy guided by the social links and the tagging activity, instead of one guided by a known query workload. Also, one difficult case in our as-you-type scenario is the one in which t_r is the initial character, following a number of already completed query terms. One possible direction for optimization in TOPKS-ASYT is to avoid revisiting users, by recording the accessed p-spaces for future reference. In short, within the memory budget, a naïve solution would be to keep these p-spaces as such (one per user). However, in order to speed-up the ranked retrieval, a more promising solution is to organize the p-spaces in a completion trie as well, which would allow us to access their entries by order of relevance.

Finally, it would be interesting to analyze our algorithm's performance when using other neighborhood-based methods for proximity, e.g., the one of [Adamic and Adar 2001] over the Tumblr or Yelp social graphs, or other path-based methods, such as SALSA [Lempel and Moran 2001], which is known to be effective in the case of directed social graphs.

REFERENCES

Lada Adamic and Eytan Adar. 2001. Friends and Neighbors on the Web. *Social Networks* 25 (2001), 211–230.

Kumaripaba Ahukorala, Alan Medlar, Kalle Ilves, and Dorota Glowacka. 2015. Balancing Exploration and Exploitation: Empirical Parameterization of Exploratory Search Systems. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM '15)*. 1703–1706.

Bahman Bahmani and Ashish Goel. 2012. Partitioned Multi-indexing: Bringing Order to Social Search. In *Proceedings of the 21st International Conference on World Wide Web (WWW '12)*. ACM, New York, NY, USA, 399–408.

Holger Bast, Christian W. Mortensen, and Ingmar Weber. 2008. Output-sensitive Autocompletion Search. *Inf. Retr.* 11, 4 (Aug. 2008), 269–286.

Holger Bast and Ingmar Weber. 2006. Type Less, Find More: Fast Autocompletion Search with a Succinct Index. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '06)*. ACM, New York, NY, USA, 364–371.

Fei Cai, Shangsong Liang, and Maarten de Rijke. 2014. Time-sensitive Personalized Query Auto-Completion. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management (CIKM '14)*. ACM, New York, NY, USA, 1599–1608.

Yi Chang, Lei Tang, Yoshiyuki Inagaki, and Yan Liu. 2014. What is Tumblr: A Statistical Overview and Comparison. *SIGKDD Explor. Newsl.* 16, 1 (Sept. 2014), 21–29.

Michael Curtiss, Iain Becker, Tudor Bosman, Sergey Doroshenko, Lucian Grijincu, Tom Jackson, Sandhya Kunnatur, Soren Lassen, Philip Pronin, Sriram Sankar, Guanghao Shen, Gintaras Woss, Chao Yang, and Ning Zhang. 2013. Unicorn: A System for Searching the Social Graph. *Proc. VLDB Endow.* 6, 11 (Aug. 2013), 1150–1161.

Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Dimitris Tsirogiannis. 2006. Answering Top-k Queries Using Views. In *Proceedings of the 32nd VLDB Conference (VLDB '06)*. 451–462.

Pavlos Fafalios and Yannis Tzitzikas. 2015. Type-ahead Exploratory Search Through Typo and Word Order Tolerant Autocompletion. *J. Web Eng.* 14, 1-2 (March 2015), 80–116.

Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal Aggregation Algorithms for Middleware. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '01)*. ACM, New York, NY, USA, 102–113.

Jianhua Feng and Guoliang Li. 2012. Efficient Fuzzy Type-Ahead Search in XML Data. *IEEE Transactions on Knowledge and Data Engineering* 24, 5 (2012), 882–895.

Wing-Kai Hon, Rahul Shah, and Jeffrey Scott Vitter. 2009. Space-Efficient Framework for Top-k String Retrieval Problems. In *50th Annual IEEE Symposium on Foundations of Computer Science (FOCS '09)*. 713–722.

Bo-June (Paul) Hsu and Giuseppe Ottaviano. 2013. Space-efficient Data Structures for Top-k Completion. In *Proceedings of the 22nd International Conference on World Wide Web (WWW '13)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 583–594.

Glen Jeh and Jennifer Widom. 2002. SimRank: A Measure of Structural-context Similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '02)*. ACM, New York, NY, USA, 538–543. DOI:http://dx.doi.org/10.1145/775047.775126

Shengyue Ji, Guoliang Li, Chen Li, and Jianhua Feng. 2009. Efficient Interactive Fuzzy Keyword Search. In *Proceedings of the 18th International Conference on World Wide Web (WWW '09)*. ACM, New York, NY, USA, 371–380.

Di Jiang, Kenneth Wai-Ting Leung, Jan Vosecky, and Wilfred Ng. 2014. Personalized Query Suggestion With Diversity Awareness. In *IEEE 30th International Conference on Data Engineering (ICDE '14)*. 400–411.

Leo Katz. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18, 1 (1953), 39–43.

Paul Lagrée, Bogdan Cautis, and Hossein Vahabi. 2015. A Network-Aware Approach for Searching As-You-Type in Social Media. In *Proceedings of the 24th ACM International Conference on Conference on Information and Knowledge Management (CIKM '15)*.

R. Lempel and S. Moran. 2001. SALSA: The Stochastic Approach for Link-structure Analysis. *ACM Trans. Inf. Syst.* 19, 2 (April 2001), 131–160.

Guoliang Li, Jianhua Feng, and Chen Li. 2013. Supporting Search-As-You-Type Using SQL in Databases. *IEEE Trans. on Knowl. and Data Eng.* 25, 2 (2013), 461–475.

Guoliang Li, Shengyue Ji, Chen Li, Jiannan Wang, and Jianhua Feng. 2010. Efficient fuzzy type-ahead search in TASTIER. In *Proceedings of the 26th International Conference on Data Engineering (ICDE '10)*. 1105–1108.

Guoliang Li, Jiannan Wang, Chen Li, and Jianhua Feng. 2012. Supporting Efficient Top-k Queries in Type-ahead Search. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '12)*. ACM, New York, NY, USA, 355–364.

Yuchen Li, Zhifeng Bao, Guoliang Li, and Kian-Lee Tan. 2015. Real time personalized search on social networks. In *31st IEEE International Conference on Data Engineering, ICDE*.

Silviu Maniu and Bogdan Cautis. 2012. Taagle: Efficient, Personalized Search in Collaborative Tagging Networks. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12)*. ACM, New York, NY, USA, 661–664.

Silviu Maniu and Bogdan Cautis. 2013a. Context-aware top-K Processing Using Views. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management (CIKM '13)*. ACM, New York, NY, USA, 1959–1968.

Silviu Maniu and Bogdan Cautis. 2013b. Network-aware Search in Social Tagging Applications: Instance Optimality Versus Efficiency. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management (CIKM '13)*. ACM, New York, NY, USA, 939–948.

Qiaozhu Mei, Dengyong Zhou, and Kenneth Church. 2008. Query Suggestion Using Hitting Time. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM '08)*. ACM, New York, NY, USA, 469–478.

Marco Pennacchiotti, Fabrizio Silvestri, Hossein Vahabi, and Rossano Venturini. 2012. Making Your Interests Follow You on Twitter. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM '12)*. ACM, New York, NY, USA, 165–174.

Michalis Potamias, Francesco Bonchi, Carlos Castillo, and Aristides Gionis. 2009. Fast Shortest Path Distance Estimation in Large Networks. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM '09)*. ACM, New York, NY, USA, 867–876.

Ralf Schenkel, Tom Crecelius, Mouna Kacimi, Sebastian Michel, Thomas Neumann, Josiane X. Parreira, and Gerhard Weikum. 2008. Efficient Top-k Querying over Social-tagging Networks. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '08)*. ACM, New York, NY, USA, 523–530.

Milad Shokouhi. 2013. Learning to Personalize Query Auto-completion. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '13)*. ACM, New York, NY, USA, 103–112.

Milad Shokouhi and Kira Radinsky. 2012. Time-sensitive Query Auto-completion. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '12)*. ACM, New York, NY, USA, 601–610.

Mohamed A. Soliman, Ihab F. Ilyas, and Shalev Ben-David. 2010. Supporting Ranking Queries on Uncertain and Incomplete Data. *The VLDB Journal* 19, 4 (Aug. 2010), 477–501.

Hossein Vahabi, Margareta Ackerman, David Loker, Ricardo Baeza-Yates, and Alejandro Lopez-Ortiz. 2013. Orthogonal Query Recommendation. In *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys '13)*. ACM, New York, NY, USA, 33–40.

Ganesh Venkataraman, Abhimanyu Lad, Viet Ha-Thuc, and Dhruv Arya. 2016. Instant Search: A Hands-on Tutorial. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*. 1211–1214.

Sen Wu, Jie Tang, and Bo Gao. 2012. Instant Social Graph Search. In *Proceedings of the 16th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II (PAKDD'12)*. Springer-Verlag, Berlin, Heidelberg, 256–267.

Sihem Amer Yahia, Michael Benedikt, Laks V. S. Lakshmanan, and Julia Stoyanovich. 2008. Efficient Network Aware Search in Collaborative Tagging Sites. *Proc. VLDB Endow.* 1, 1 (Aug. 2008), 710–721.

Aston Zhang, Amit Goyal, Weize Kong, Hongbo Deng, Anlei Dong, Yi Chang, Carl A. Gunter, and Jiawei Han. 2015. adaQAC: Adaptive Query Auto-Completion via Implicit Negative Feedback. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '15)*. 143–152.

Ruicheng Zhong, Ju Fan, Guoliang Li, Kian-Lee Tan, and Lizhu Zhou. 2012. Location-aware Instant Search. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM '12)*. ACM, New York, NY, USA, 385–394. DOI:http://dx.doi.org/10.1145/2396761.2396812

Justin Zobel and Alistair Moffat. 2006. Inverted Files for Text Search Engines. *ACM Comput. Surv.* 38, 2, Article 6 (July 2006).