Context-Aware Top-k Processing using Views

Silviu Maniu Bogdan Cautis
Institut Mines-Télécom, Télécom ParisTech - CNRS LTCI
first.last@telecom-paristech.fr

Abstract

Search applications in which queries are dependent on their context are becoming increasingly relevant in today's online applications. For example, the context may be the location of the user in *location-aware search* or the social network of the query initiator in *social-aware search*. Processing such queries efficiently is inherently difficult, and requires techniques that go beyond the existing, context-agnostic ones. A promising direction for efficient, online answering – especially in the case of top-k queries – is to materialize and exploit previous query results (views).

We consider in this paper context-aware query optimization based on views, focusing on two important sub-problems. First, handling the possible differences in context between the various views and an input query leads to view results having uncertain scores, i.e., score ranges valid for the new context. As a consequence, current top-k algorithms are no longer directly applicable and need to be adapted to handle such uncertainty in object scores. Second, adapted view selection techniques are needed, which can leverage both the descriptions of queries and statistics over their results. We present algorithms that address these two problems, and illustrate their practical use in two important application scenarios: location-aware search and social-aware search. We validate our approaches via extensive experiments, using both synthetic and real-world datasets.

Keywords: social search, spatial search, context-aware search, top-k algorithms, information retrieval

1 Introduction

Retrieving the k best data objects for a given query, under a certain scoring model, is one of the most common problems in database systems and on Web. In many applications, and in particular in current Web search engines, tens of thousands of queries per second need to be answered over massive amounts of data. Significant research effort has been put into addressing the performance of top-k processing, towards optimal algorithms – such as TA and NRA [7, 10] – or highly-efficient data structures [19] (e.g., inverted lists). In recent research, the use of *pre-computed results* (also called *views*) has been identified as a promising avenue for improving efficiency [11, 6].

At the same time, with the advent of location-aware devices, geo-tagging, bookmarking applications, or online social applications, as a way to improve the result quality and the user experience, new kinds of top-k search applications are emerging, which can be simply described as *context-aware*. The context of a query may represent the geographic *location* where the query was issued or the *social identity* – within a social network – of the user who issued it. More generally, it could represent certain score parameters that can be defined or personalized at query time. For example, a query for *top-class vegetarian restaurants* should not give the same results if issued in *Paris* or in *Istanbul*, as it should not give the same results if issued within a social community of culinary reviewers or within a student community.

Unsurprisingly, taking into account a query context in top-k processing represents a new source of complexity, and many of the common approaches employed in context-agnostic scenarios need to be revisited [5, 14, 12]. Now, query processing usually entails an exploration of a "neighborhood" space for the closest or most relevant objects, which is often interleaved with some of the classic, context independent top-k processing steps, such as scans over inverted lists.

Consequently, materializing and exploiting in searches the results of previous queries can play an even more important role for efficient, online processing of queries with context. However, in this direction, a broader view-based answering problem than in the context-agnostic setting needs to be addressed, in which the cached results are modeled as unranked lists of objects having only *uncertain scores* or *score ranges*, instead of exact scores. The rationale is that, even when the cached results in views do have exact scores with respect to one context, we should expect these to evolve into score ranges when a *context transposition* is necessary. For example, answers to the previous query, for the *Paris* context, may be useful – but only to a certain extent – when the same query is issued in a nearby *Versailles* context, as one has to adapt the scores of restaurants from the parisian perspective to the versaillaise one; this, inherently, introduces uncertainty.

The potential impact of view-based algorithms that can cope with such uncertainty is highly relevant but not limited to the context-aware setting. Indeed, even when queries are not parameterized by a context, some of the most performant algorithms, such as NRA [7] can support early-termination and output unranked results with only score ranges (instead of a precise ranking).

The general goal of this study is to enable efficient context-aware top-k retrieval through techniques that exploit *exclusively* the views. The rationale for this is that in many practical applications, access methods may be extensively optimized for views, the size of cached results may be much less important than the one of the complete data (e.g., of the inverted lists), and view results (pre-computed for groups of attributes) may be much more informative towards finding the result

for the input query. For instance, a user may go through a sequence of query reformulations, for which result caching may be highly beneficial. View results may even be bound to main-memory, in certain scenarios.

Our contributions We formalize and study in this paper the problem of context-aware top-k processing based on possibly uncertain precomputed results, in the form of *views* over the data.

We start by investigating top-k processing *after the context transposition* has been performed, for a given input query and its context. The problem of answering such top-k queries using *only* the information in views, inevitably, requires an adaptation to the fact that these views may now offer objects having uncertain scores. Consequently, there might exist view instances from which an exact top-k cannot be extracted with full confidence. When this is the case, it would be unsatisfactory to simply refute the input query, or to consider alternative, more expensive execution plans (e.g., by going through the per-attribute lists). Instead, it would be preferable to provide a *most informative answer*, in terms of (i) objects G that are guaranteed to be in the top-k result, and (ii) objects P that may appear in the top-k result.

We formalize this query semantics and describe two adaptations of TA and NRA, SR-TA and SR-NRA. They support precomputed lists with *score ranges* and the above described query semantics and are *sound* and *complete*, i.e., they output the (G, P)-answer. Intuitively, they implement the corroboration principle illustrated before, based on a *linear programming* formulation.

Given that in many applications the set of views may be very large – think of social applications in which many users may have pre-computed results – we also consider optimizations for SR-TA and SR-NRA, based on *selecting some (few) most promising views*. Obviously, with fewer views, the most informative answer (G, P) may no longer be reached, and we are in general presented a trade-off between the number of selected views – which determines the cost of the top-k algorithms SR-NRA and SR-TA – and the "quality" of the result (a distance with respect to the most informative answer given by all the views). Importantly, we also show that SR-NRA and SR-TA, when selecting views, are *complete* and *instance optimal* for an important family of view specifications.

Complementing our top-k retrieval through view selection, we also show how a final refinement step allows us to reach the most informative result.

As a last level of service that can be provided to users, we then consider a sampling-based approach by which, from the most informative result, a probabilistic interpretation can also lead to a *most likely top-k* answer to the input query.

Extensive experiments on both synthetic and real-world datasets illustrate the potential of our techniques – enabling high-precision retrieval and important running-time savings. More generally, they illustrate the potential of top-k query optimization based on cached results in a wide range of applications.

2 Formal setting and problems

Context-aware score model. We assume a finite collection of objects $\mathcal{O}$ and a countable collection of attributes $\mathcal{T}$. Under a given *context parameter* $\mathcal{C}$ – an application-dependent notion – objects o

are associated to certain attributes t, by an object-attribute score function $sc(o, t \mid \mathcal{C})$.

Under a context $\mathcal{C}$, a query Q consists of a set of attributes $\{t_1, \ldots, t_n\}$; its answer is given by objects $o \in \mathcal{O}$ having the highest scores $sc(o, Q \mid \mathcal{C})$, computed via a monotone aggregation function h (e.g., `sum`, `max`, `avg`) over the object-attribute scores:

$$sc(o, Q \mid \mathcal{C}) = h(sc(o, t_1 \mid \mathcal{C}), \ldots, sc(o, t_n \mid \mathcal{C})).$$

We can formalize the top-k retrieval problem as follows:

Problem 1. *Given a query $Q = \{t_1, \ldots, t_n\} \subset \mathcal{T}$, a context $\mathcal{C}$, an integer k, and a score model specification (sc, h), retrieve the k objects $o \in \mathcal{O}$ having the highest scores $sc(o, Q \mid \mathcal{C})$.*

In certain applications, the context may always be empty or may simply be ignored in the sc scores, and, when necessary, we indicate this in our notation by the '$\perp$' context. We use $sc(o, Q)$ as short notation for $sc(o, Q \mid \perp)$.

Threshold algorithms. We revisit in this paper the class of early termination top-k algorithms known as *threshold algorithms*. These algorithms, applicable in a context-agnostic setting, find the top-k objects for an input query Q by scanning sequentially (for each attribute) and in parallel (for the entire attribute set of Q), relevant per-attribute lists that are ordered descending by sc values – with inverted lists being a notable example – denoted in the following $L(t)$, as the list for attribute t. During a run, they maintain a set D of already encountered candidate objects o, bookkeeping for each candidate the following values:

1. an upper-bound on $sc(o, Q)$, the best possible score that may still be obtained for o, denoted hereafter $bsc(o, Q)$,
2. a lower-bound on $sc(o, Q)$, the worst possible score, denoted hereafter $wsc(o, Q)$,

with the objects being ordered in D by their worst scores.

At each iteration, or at certain intervals, threshold algorithms may refine these bounds and compare the worst score of the kth object in D, $wsc(D[k], Q)$, with the best possible score of either (i) objects o in D outside the top k, $bsc(o, Q)$, or (ii) not yet encountered objects, denoted $bsc(*, Q)$.

When both these best scores are not greater than the worst score of $D[k]$, the run can terminate, outputting the objects $D[1], \ldots, D[k]$ as the final top-k.

A key difference between the various threshold algorithms, and in particular between TA and NRA, resides in the way they are allowed to access the per-attribute lists. TA is allowed random accesses to lists, as soon as an object o has been encountered while sequentially accessing one list among the $|Q|$ relevant ones. These random accesses can complete the scores of objects o from a guaranteed range to an exact value. In comparison, NRA is not allowed to use random accesses in the per-attribute lists, but only sequential ones, and each object o in the final top-k may only be given a score range, $[wsc(o, Q), bsc(o, Q)]$. (Various hybrid algorithms with respect to TA and NRA are also possible and have been extensively studied in the literature.)

$V_1(\{a\}, \perp)$			$V_2(\{c\}, \perp)$			$V_3(\{a, b\}, \perp)$			$V_4(\{b, c\}, \perp)$		
o	**ws**	**bs**	**o**	**ws**	**bs**	**o**	**ws**	**bs**	**o**	**ws**	**bs**
o3	7	8	o3	8	8	o5	16	16	o5	11	11
o5	6	7	o4	3	7	o6	10	10	o3	10	11
o6	4	4	o6	2	4	o3	10	10	o6	9	11
o7	3	5	o10	2	3	o10	6	9	o10	8	9
o9	3	4	o7	2	2	o2	6	6	o2	6	7
o10	1	3	o8	1	3	o7	6	6	o1	5	7
o2	1	2	o9	1	2	o1	5	8	o7	4	4
o1	1	1	o5	1	1	o9	4	5	o4	3	8
$*$	0	1	$*$	0	1	$*$	0	4	$*$	0	3

Table 1: An example set $\mathcal{V}$ of views.

TA was shown to be *instance optimal*[1] among algorithms that do not make "wild guesses" or probabilistic choices. Within this same class of algorithms, NRA was shown to be instance optimal for algorithms in which only *sequential* accesses are allowed.[2]

Views and precomputed results. We extend the classic top-k retrieval setting of TA/NRA by assuming access to precomputed query results, called in the following *views*. Each view V is assumed to have two components: (i) a *definition*, $def(V)$, which is a pair query-context $def(V) = (Q^V, \mathcal{C}^V)$ and (ii) a set $ans(V)$ of triples (o_i, wsc_i, bsc_i), representing the *answer* to query Q^V under context $\mathcal{C}^V$. Each such triple says that object o_i has a score $sc(o_i, Q^V \mid \mathcal{C}^V)$ within the range $[wsc_i, bsc_i]$.

Since we are dealing with cached query results, all objects not appearing in $ans(V)$ – represented explicitly in $ans(V)$, to simplify presentation, by one final *wildcard* $*$ *object* – have with respect to query Q^V and context $\mathcal{C}^V$ a worst score of $wsc_* = 0$ and a best possible score of either $bsc_* = \min\{wsc_i \mid (o_i, wsc_i, bsc_i) \in ans(V)\}$, if V's result is complete, in the sense that enough objects had a non-zero score w.r.t. Q^V, or otherwise 0.

Context transposition. Intuitively, when a view V and the to-be-answered query Q do not share the same context, a transposition of the exact scores or score ranges in $ans(V)$ is necessary, in order to obtain valid ranges for $sc(o_i, Q^V \mid \mathcal{C})$ from those for $sc(o_i, Q^V \mid \mathcal{C}^V)$. In particular, in the case of spatial or social search, this transformation will inevitably yield a coarser score range. We will detail the specific operation of context transposition for these two application scenarios in Section 6.

Exploiting views. Given an input query Q and a context $\mathcal{C}$, from a set of views $\mathcal{V}$ sharing the same context – as in $def(V) = (\ldots, \mathcal{C})$ – a first opportunity that is raised by the ability to cache results is to compute for objects $o \in \mathcal{O}$ tighter lower and upper bounds over $sc(o, Q \mid \mathcal{C}))$. This may be useful in threshold algorithms, as a way to refine score ranges. We formalize this task next.

Problem 2. *Given a query $Q = \{t_1, \ldots, t_n\} \subset \mathcal{T}$, a context $\mathcal{C}$, an integer k, a score model specification (sc, h) and a set of views $\mathcal{V}$ sharing the same context with Q, given an object $o \in \mathcal{O}$, compute the tightest lower and upper bounds on $sc(o, Q \mid \mathcal{C})$ from the information in $\mathcal{V}$.*

[1] As good as any other algorithm from its class – within a constant factor – over all possible inputs.

[2] This is an important family of algorithms for performance, given that random accesses can be orders of magnitude more costly.

In this paper, consistent with the most common ranking models for context-aware search, we will assume that the aggregation function h is summation. Under this assumption, Problem 2 could be modeled straightforwardly by the following mathematical program, whose variables are given in bold:

$$\min \sum_{t_i \in Q} \mathbf{sc}(\mathbf{o}, \mathbf{t_i} \mid \mathcal{C}) \tag{2.1}$$

$$\max \sum_{t_i \in Q} \mathbf{sc}(\mathbf{o}, \mathbf{t_i} \mid \mathcal{C}) \tag{2.2}$$

$$wsc \leq \sum_{t_j \in Q^V} \mathbf{sc}(\mathbf{o}, \mathbf{t_j} \mid \mathcal{C}),\ \forall V \in \mathcal{V}\ s.t.\ (o, wsc, bsc) \in ans(V)$$

$$\sum_{t_j \in Q^V} \mathbf{sc}(\mathbf{o}, \mathbf{t_j} \mid \mathcal{C}) \leq bsc,\ \forall V \in \mathcal{V}\ s.t.\ (o, wsc, bsc) \in ans(V)$$

$$\mathbf{sc}(\mathbf{o}, \mathbf{t_l} \mid \mathcal{C}) \geq 0, \forall t_l \in \mathcal{T}$$

Example 1. *Let us consider the views in Table 1. We have access to the results of four views, defined by the sets of attributes $\{a\}$, $\{c\}$, $\{a, b\}$ and $\{b, c\}$. We assume the empty context $\mathcal{C} = \perp$ for the views and for the to-be-answered query, which is $Q = \{a, b, c\}$. Considering $o6$, for example, we know that:*

$$\begin{aligned}
sc(o6, \{a\}) &\geq 4 && (V_1)\\
sc(o6, \{c\}) &\geq 2 && (V_2)\\
sc(o6, \{a\}) + sc(o6, \{b\}) &\geq 10 && (V_3)\\
sc(o6, \{b\}) + sc(o6, \{c\}) &\geq 9 && (V_4)\\
sc(o6, \{a\}) &\leq 4 && (V_1)\\
sc(o6, \{c\}) &\leq 4 && (V_2)\\
sc(o6, \{a\}) + sc(o6, \{b\}) &\leq 10 && (V_3)\\
sc(o6, \{b\}) + sc(o6, \{c\}) &\leq 11 && (V_4)
\end{aligned}$$

Then, the lower bound on $sc(o6, Q)$ is obtained as also

$$wsc(o6) = \min(sc(o6, \{a\}) + sc(o6, \{b\}) + sc(o6, \{c\})) = 13$$

by combining the worst scores of V_1 and V_4. Similarly, the upper bound on $sc(o6, Q)$ is obtained as

$$bsc(o6, Q) = \max(sc(o6, \{a\}) + sc(o6, \{b\}) + sc(o6, \{c\})) = 14$$

by combining the best scores of V_2 and V_3.

We now formulate the problem of answering input top-k queries Q using *only* the information in views, whose semantics needs to be adapted to the fact that views may offer only a partial image of the data. When an exact top-k cannot be extracted with full confidence, a *most informative result* would consist of two disjunctive, possibly-empty sets of objects from those appearing in $\mathcal{V}$'s answer:

- a set of all the objects guaranteed to be in the top-k for Q
- a set of all objects that may also be in the top-k for Q.

Problem 2 provides a way to properly define and identify objects of the former kind – the *guaranteed ones* – as the objects o_x for which

$$\min \sum_{t_i \in Q} sc(o_x, t_i \mid \mathcal{C}) \geq \max \sum_{t_i \in Q} sc(*, t_i \mid \mathcal{C}) \tag{2.3}$$

and at most $k-1$ objects o_y can be found such that

$$\min \sum_{t_i \in Q} sc(o_x, t_i \mid \mathcal{C}) < \max \sum_{t_i \in Q} sc(o_y, t_i \mid \mathcal{C}). \tag{2.4}$$

Similarly, we can identify objects of the latter kind – the *possible ones* – as the objects o_x that are not guaranteed and for which at most $k-1$ objects o_y can be found such that

$$\min \sum_{t_i \in Q} sc(o_y, t_i \mid \mathcal{C}) > \max \sum_{t_i \in Q} sc(o_x, t_i \mid \mathcal{C}). \tag{2.5}$$

We formalize the *top-k retrieval problem using views* as follows.

Problem 3. *Given a query $Q = \{t_1, \ldots, t_n\} \subset \mathcal{T}$, a context $\mathcal{C}$, an integer k, and a score model specification (sc, h), given a set of views $\mathcal{V}$ sharing the same context with Q, retrieve from $\mathcal{V}$ a* most informative answer *of the form (G, P), with*

- *$G \subset \mathcal{O}$ consisting of all* guaranteed objects *(as in Eq. (2.3) and (2.4), when h is summation); they must be among those with the k highest scores for Q and $\mathcal{C}$.*
- *and $P \subset \mathcal{O}$ consisting of all* possible objects *outside G (as in Eq. (2.5), when h is summation); they may be among those with the k highest scores for Q and $\mathcal{C}$, i.e., there exist data instances where these appear in the top-k.*

In order to solve Problem 3, a naïve computation of upper and lower bounds for all objects o appearing in the views would suffice, but would undoubtedly be too costly in practice. Instead, we show in Section 3 how we can solve Problem 3 in the style of threshold algorithms, by extending NRA and TA.

Over any data instance, the exact top-k can be thought of as the set G plus the top-k' items from P, for $k' = k - |G|$. To give a *most likely result*, in a probabilistic sense, based on the G and P object sets, we discuss in Section 3 possible approaches for estimating the probability of possible top-k' sets from P.

Going further, even when the most promising candidate objects are considered first in SR-TA or SR-NRA, their corresponding instances of the mathematical programs in Eq. (2.1) and Eq. (2.2) may still be too expensive to compute in practice (even when we are dealing with LPs, as in Example 1): the set of views may be too large – potentially of the order $2^{|\mathcal{T}|}$ – and each

view contributes one constraint in the program. In our best-effort approach, which would first *select some (few) most promising views* $\tilde{\mathcal{V}} \subset \mathcal{V}$ for the input query (Section 4), we are presented a trade-off between the size of the subset $\tilde{\mathcal{V}}$ – which determines the cost of the top-k algorithms SR-NRA and SR-TA – and the "quality" of the result, namely its distance with respect to the most informative answer given by all the views. We quantify the distance between the most informative result by $\tilde{\mathcal{V}}$, denoted $(\tilde{G}, \tilde{P})$, and the most informative answer (G, P) by $\mathcal{V}$ as the difference in the number of possible top-k combinations:

$$\Delta = \binom{|\tilde{P}|}{k - |\tilde{G}|} - \binom{|P|}{k - |G|}. \tag{2.6}$$

We also show in Section 4 how a final refinement step over $(\tilde{G}, \tilde{P})$, based on random accesses in the entire $\mathcal{V}$ set, allows us to reach $\Delta = 0$, i.e., the most informative result by $\mathcal{V}$.

3 Threshold algorithms

We start this section by presenting our adaptation of TA, called SR-TA, which can be applied when the input lists consist of objects with score ranges; SR-TA will allow us to solve Problem 3.

Each of the input lists are assumed to be available in two copies, one ordered descending by the score lower-bound and one ordered descending by the score upper-bound. SR-TA will read sequentially in round-robin manner from the former group of lists and, similar to TA, maintains a candidate set D of the objects encountered during the run. At each moment, the read heads of the latter group of lists must give objects that are not yet in D (unseen objects), and sequential accesses are performed in SR-TA whenever necessary in order to maintain this configuration.

D is also ordered descending by the score lower-bounds. The algorithm stops when the score of any of the unseen objects – the threshold τ – cannot be greater than the one of the kth object in the candidate set D.

In our setting, the threshold τ is obtained as the solution of the following mathematical program, taking into account from each view V the score upper-bound of objects from $ans(V) - D$:

$$\begin{aligned} &\tau = \max \sum_{t_i \in Q} \mathbf{sc}(\mathbf{o}, \mathbf{t_i} \mid \mathcal{C}) \\ &\sum_{t_j \in Q^V} \mathbf{sc}(\mathbf{o}, \mathbf{t_j} \mid \mathcal{C}) \leq max(bsc_i),\ \forall V \in \mathcal{V},\ o_i \notin D\ s.t. \\ &\qquad\qquad\qquad\qquad\qquad\qquad (o_i, wsc_i, bsc_i) \in ans(V) \\ &\mathbf{sc}(\mathbf{o}, \mathbf{t_l} \mid \mathcal{C}) \geq 0, \forall t_l \in \mathcal{T} \end{aligned} \tag{3.1}$$

One can note that when (i) we have only views that give answers to singleton queries, and (ii) the $wsc_i = bsc_i$ for each object o_i (i.e., the lists contain exact scores), we are in the setting of the TA family of algorithms over inverted list inputs. Relaxing condition (i), we have the setting of top-k answering using views investigated in [11, 6]. Both these settings and their corresponding algorithms can guarantee that, at termination, the exact top-k is returned.

Our more general setting, however, cannot provide such guarantees, as witnessed by the following example.

Algorithm 1: SR-TA$(Q, k, \mathcal{V})$

Require: query Q, size k, views $\mathcal{V}$

1: $D = \emptyset$
2: **loop**
3: **for** each view $V \in \mathcal{V}$ in turn **do**
4: $(o_i, wsc_i, bsc_i) \leftarrow$ next tuple by sequential access in V
5: read by random-accesses all other lists $V' \in \mathcal{V}$ for tuples (o_j, wsc_j, bsc_j) s.t. $o_i = o_j$
6: $wsc \leftarrow$ solution to the MP in Eq. (2.1) for o_i
7: $bsc \leftarrow$ solution to the MP in Eq. (2.2) for o_i
8: add the tuple (o_i, wsc, bsc) to D
9: **end for**
10: $\tau \leftarrow$ the solution to the MP in Eq. (3.1)
11: $wsc_t \leftarrow$ lower-bound score of kth candidate in D
12: **if** $\tau \leq wsc_t^{(D)}$ **then**
13: **break**
14: **end if**
15: **end loop**
16: $\{G, P\}$=PARTITION(D, k)
17: **return** G, P

Example 2. *Let us revisit Example 1, for the top-5 query $Q = \{a, b, c\}$. We will not detail the complete run of the algorithm on this example, instead showing what happens at termination. The algorithm stops at the 6th iteration. The threshold value is either obtained by combining the best scores in $V1$ and $V4$ of the unseen (not in D) item $o1$, or by combining the best score in $V2$ of $o8$ and the best score in $V3$ of $o1$. Both result in $\tau = 8$. The worst score of the 5th item, $o7$, is also 8, enabling termination. This ensures that all the possible candidates for top-k are already present in the list D (see Table 2). Within this candidate list, there does not exist a combination of 5 objects that represents the top-k and, instead, we can only divide D into three sets:*

1. *the set $G = \{o3, o5, o6, o10\}$ of guaranteed result objects,*
2. *the set $P = \{o7, o4\}$ of possible result objects,*
3. *the remaining objects: $\{o2, o9\}$.*

Algorithm 1 details SR-TA. Its general flow is similar to the one of TA, with the notable addition of the generalized computation of bounds and of the threshold value.

We now discuss our adaptation of NRA, called SR-NRA. Now, the exclusively sequential nature of accesses to views means that the per-view scores will only be partially filled (the random accesses in line 4 of SR-TA are no longer possible).

obj	o3	o5	o6	o10	o7	o2	o9	o4
wsc	18	17	13	9	8	7	5	3
bsc	18	17	14	12	8	7	7	9

Table 2: Candidates (D) at termination, for Q=$\{a, b, c\}$, k=5.

At any moment in the run of SR-NRA, $seen(o, \mathcal{V}) \subseteq \mathcal{V}$ gives the views in which o has been encountered already through sequential accesses. We say that an object is *fully known* if $seen(o, \mathcal{V}) = \mathcal{V}$, and *partially known* otherwise. Then, for views $V \in seen(o, \mathcal{V})$ we keep the same constraints as in the MPs (2.1), (2.2). For each view $V \notin seen(o, \mathcal{V})$, we adjust the corresponding constraint as

$$0 \leq \sum_{t_j \in Q^V} sc(o, t_j|\mathcal{C}) \leq \max\{bsc_i | (o_i, bsc_i, wsc_i) \in V, o_i \notin D\} \tag{3.2}$$

The termination conditions need to keep track, besides the threshold value, of the maximum upper-bound score of partially known objects that not in the current top-k of D, denoted bsc_{rest}. Objects that are fully known are ignored in this estimate, since their scores are fully filled and they might be candidates for P.

Partition for most informative result. Once the main loop of SR-TA or SR-NRA terminates, candidates D are passed as input to a sub-routine whose role is to partition it into sets G and P (line 14 in SR-TA, line 25 in SR-NRA). Algorithm 2 details this step: for each object o in D we test the conditions of Eq. (2.3), (2.4), (2.5).

Algorithm 2: PARTITION(D, k)

Require: candidate list D, parameter k

1: $G \leftarrow \emptyset$ *the objects guaranteed to be in the top-k*
2: $P \leftarrow \emptyset$ *the objects that might enter the top-k*
3: **for** each tuple $(o, bsc, wsc) \in D$, $o \neq *$ **do**
4: $\quad x \leftarrow |\{(o', bsc', wsc') \in D \mid o' \neq o,\ bsc' > wsc\}|$
5: $\quad wsc_t \leftarrow$ lower-bound score of kth candidate in D
6: $\quad$ **if** $x \leq k$ and for $(*, wsc_*, bsc_*) \in D, bsc_* \leq wsc$ **then**
7: $\quad\quad$ add o to G
8: $\quad$ **else if** $bsc > wsc_t$ **then**
9: $\quad\quad$ add o to P
10: $\quad$ **end if**
11: **end for**
12: **return** G, P

At the termination of both SR-TA and SR-NRA, we are guaranteed that G and P are *sound and complete*, in the following sense:

Property 1. *An object o is in the output set G of* PARTITION(D, k) *iff in all possible data instances o is the top-k for Q, $\mathcal{C}$.*

An object o is in the output set P of PARTITION(D, k) *iff in at least one possible data instance o is in the top-k for Q, $\mathcal{C}$.*

Note that the size of G is at most k, while the one of P is at most $|\mathcal{O}|$, hence the need for completeness, maximizing $|G|$ and minimizing $|P|$.

Extracting a Probable Top-k' in P As discussed previously, the actual (inaccessible) top-k answer for the input query could be seen as being composed of two parts: the guaranteed objects G plus a top k' over P, for $k' = k - |G|$. By definition, G and P give the most informative certain result that can be obtained from the views: there can be no deterministic way to compute a certain top-k' over the P objects, nor a way to further prune the search space towards a more refined P set.

Therefore, we can only hope to improve the quality of the result by a more detailed *probabilistic* description of the result, in which a most likely top-k could be identified from G and P. Since for each object in P we have a lower and upper bound on its exact score, let us assume a known probability-density function (e.g, uniform one) for scores within the known bounds. Based on this, we can reason about the likelihood of a top-k' selection over P.

A naïve way to obtain the most likely top-k' would be the following: enumerate all possible subsets of P of size k', and compute for each the probability of being the top-k'. Each of these $\binom{|P|}{k'}$ probability values can be easily obtained once we have for each pair of objects $o_1, o_2 \in P$ the probability $Pr(o_1 > o_2)$. A much more efficient algorithm than the naïve enumeration is to adapt to our setting the sampling-based approach of [15], which computes top-k answers over uncertain data, namely ranked object list with score ranges and probability-density functions over them.

We describe a tractable approach for estimating the most likely top-k' over a set of triples (o_i, wsc_i, bsc_i), under the assumption that sampling can be done in polynomial-time as well, for uniform distribution. We use an encoding-decoding pair of functions that map sets of objects to numerical keys, and vice-versa: $key = encode(S)$ is the key representing the set S, and $S = decode(key)$ gives the opposite mapping.

We proceed as follows. We first initialize a hash table T for the domain of keys (range of $encode$). For a given number of sampling rounds, at each round l we go through the objects of P and generate for each a score based on its range; we then order the objects based on these scores into a list P_l ($sample\_scores$ subroutine). We obtain through $encode$ the key for the set consisting of the top k' objects in P_l, and we increment the value corresponding to that key in T. At termination, we return the decoding of the key having the highest count in T.

4 View Selection

We consider now the view selection problem, which may improve the performance of our threshold algorithms SR-NRA and SR-TA, possibly at the risk of yielding results that are less accurate. To address this issue, we discuss at the end of this section how results obtained through view selection

can be refined to the most informative one. *Throughout this section, we remain in the setting where the query and views are assumed to have the same context.*

We argue first that view selection comes as a natural perspective in the computation of score bounds. Recall that, for a given object $o \in O$, Problem 2 could be modeled straightforwardly by the mathematical programs (2.1) and (2.2). Put otherwise, we have as the dual of the minimization problem 2.1 the following packing LP:

$$\max \sum_{\substack{i=1,\\ (o,wsc,..)\in ans(V_i)}}^{|\mathcal{V}|} wsc \times l_i \; s.t. \qquad \sum_{t\in Q^{V_j}} l_j \leq 1, \forall t \in Q, \sum_{t\in Q^{V_j}} l_j = 0, \forall t \notin Q \tag{4.1}$$

and we have as the dual the maximization problem (2.2) the following covering LP:

$$\min \sum_{\substack{i=1,\\ (o,..,bsc)\in ans(V_i)}}^{|\mathcal{V}|} bsc \times u_i \; s.t. \qquad \sum_{t\in Q^{V_j}} u_j \geq 1, \forall t \in Q, \sum_{t\in Q^{V_j}} u_j = 0, \forall t \notin Q \tag{4.2}$$

Based on the programs (4.1) and (4.2), for each object o, in order to obtain its most refined bounds, we would need to first *fractionally* select views from $\mathcal{V}$ – as opposed to *integral* selection – such that the linear combinations of o's scores with the coefficients u_i and l_i are optimal. In other words, for computing the worst score or best score of each object, it would suffice to select and take into account only the views $V_i \in \mathcal{V}$ such that (i) $l_i \neq 0$, for worst scores, or (ii) $u_i \neq 0$, for best scores.[3]

Solving the LPs (4.1) and (4.2) for each object, as a means to select only the useful views, would obviously be as expensive as solving directly the MPs (2.1) and 2.2. Instead, it would be preferable to solve these LPs and select some most relevant views *independently of any object*, i.e., only once, before the run of the threshold algorithm. Instead of per-object wsc and bsc values, in an approximate version of the two LPs, each view V_i could be represented by two unique values, $wsc(V)$ and $bsc(V)$. Our optimization problems would then simplify as follows:

$$\max \sum_{i=1}^{|\mathcal{V}|} wsc(V_i) \times l_i \; s.t. \qquad \sum_{t\in Q^{V_j}} l_j \leq 1, \forall t \in Q, \sum_{t\in Q^{V_j}} l_j = 0, \forall t \notin Q \tag{4.3}$$

$$\min \sum_{i=1}^{|\mathcal{V}|} bsc(V_i) \times u_i \; s.t. \qquad \sum_{t\in Q^{V_j}} u_j \geq 1, \forall t \in Q, \sum_{t\in Q^{V_j}} u_j = 0, \forall t \notin Q \tag{4.4}$$

[3]Restricting the domain of the u and l values to integers would lead to an NP-hard view selection problem. More precisely, Eq. (4.2) would reduce to an instance of the weighted set cover problem, and Eq. (4.1) would reduce to an instance of the k-dimensional perfect matching problem (where $k = \max(|Q^{(V)}|), \forall V \in \mathcal{V}$). In our setting, however, the restriction to the integer domain is not necessary, and there exist tractable methods for efficiently solving the above LPs in their fractional form.

and this would enable us to select the "good" views in the initialization step of the top-k algorithm, those participating to the computation of the optimal, i.e., views having *non-zero* u and l coefficients.

Furthermore, for each object o encountered in the run of Algorithms SR-TA and SR-NRA, we can now replace Eq. (2.1) and (2.2) (lines 5-6 in SR-TA) by the following estimates that use only the selected views $\tilde{\mathcal{V}}$:

$$\widetilde{wsc} = \sum_{\substack{i=1 \\ (o,wsc,..)\in ans(V_i)}}^{|\tilde{\mathcal{V}}|} wsc \times l_i \; ; \; \widetilde{bsc} = \sum_{\substack{i=1 \\ (o,..,bsc)\in ans(V_i)}}^{|\tilde{\mathcal{V}}'|} bsc \times u_i$$

This is possible since, by the duality property, we are guaranteed that the feasible solutions for Eq. (4.3), (4.4) represent safe bounds for o's scores, i.e., $\widetilde{wsc} \leq wsc$ and $\widetilde{bsc} \geq bsc$. We can similarly simplify Eq. (3.1), for the threshold value (for line 8 in SR-TA).

Candidates for $wsc(V)$ and $bsc(V)$. We follow the described approach – approximating view selection – in two distinct ways.

First, per-view score bounds $wsc(V)$ and $bsc(V)$ could be based solely on the view's definition Q^V, and we experimented in this paper with bounds that are defined as $wsc(V) = bsc(V) = |Q^V|$. for each $V \in \mathcal{V}$. The intuition for this choice is that object scores in a view V are proportional to the number of attributes in Q^V.

Second, we consider and experiment with in Section 7 two natural per-view measures that are based on the views' answers: (i) the average value of scores, and (ii) the maximum value of scores.

Retrieving (G, P) after view selection We now discuss how the most informative result (G, P) – that can be obtained from the complete set of views $\mathcal{V}$ – can still be retrieved by refining a result $(\tilde{G}, \tilde{P})$ obtained on a selection of views $\tilde{\mathcal{V}}$. We only need to adopt the following modifications in instances of SR-TA or SR-NRA running over a selection of views:

1. when the main loop terminates, compute the optimal bounds for all objects in $\tilde{P}$ by random-accessing their scores in all the views in $\mathcal{V}$,

2. run for a second time the partition subroutine.

It can be easily shown that, in this way, we obtain the most informative result, i.e., we reach $\Delta = 0$. Therefore, the "bulk" of the work could be done only on a selection of views and its result, potentially few candidate objects, could just be refined at the end using the complete $\mathcal{V}$. We describe in Section 7 the impact of this optimization on the running time of SR-TA and SR-NRA.

To summarize, we have described two variants of SR-TA and SR-NRA: without view selection, denoted SR-TA^{nosel} and SR-NRA^{nosel}, and with view selection, denoted SR-TA^{sel} and SR-NRA^{sel}. For the view selection variant, our notation convention will be to replace the sel superscript by a def, max or avg one, depending on the selection method being used.

5 Formal Guarantees

We study in this section the formal properties of our algorithms, focusing on *instance optimality*.

Let $\boldsymbol{A}$ be the class of algorithms, including SR-TA and SR-NRA, that deterministically output the sound and complete sets P and G, and do not make "wild guesses". For a given set of views $\mathcal{V}$, we denote by $D(\mathcal{V})$ the class of all instances of answers in those views, i.e., $ans(V), V \in \mathcal{V}$.

Given two algorithms $A_1 \in \boldsymbol{A}$ and $A_2 \in \boldsymbol{A}$, we write $A_1 \preceq A_2$ iff, for all sets of views $\mathcal{V}$, A_2 is guaranteed to cost at least as much as A_1 – in terms of I/O accesses (sequential, random or a linear combination of the two) – over all instances in $D(\mathcal{V})$. Conversely, we write $A_1 \not\preceq A_2$ iff there exists at least one view set $\mathcal{V}$ and an instance in $D(\mathcal{V})$ over which A_2 costs less than A_1. We say that an algorithm $A \in \boldsymbol{A}$ is *instance optimal over* $\boldsymbol{A}$ iff $A \preceq B, \forall B \in \boldsymbol{A}$.

We first consider the question whether one of the two variants of SR-TA or SR-NRA is guaranteed to perform better that the other, for all views and answers. The answer to this question is far from obvious: on one hand, SR-TA^{sel} or SR-NRA^{sel} should use fewer views to compute the P and G sets, but they might either go too deep in the selected views or might need additional accesses in other views (see Section 4); on the other hand, SR-TA^{nosel} or SR-NRA^{nosel} may go through views that are useless for deriving optimal bounds. We can prove the following:

Lemma 1. SR-NRA^{sel} $\not\preceq$ SR-NRA^{nosel} $\not\preceq$ SR-NRA^{sel} *and* SR-TA^{sel} $\not\preceq$ SR-TA^{nosel} $\not\preceq$ SR-TA^{sel}.

Lemma 1 tells us that neither of the two variants of SR-TA or SR-NRA can be instance optimal for all possible sets $\mathcal{V}$. However,
we describe next a restricted class of views for which: (i) no refinement step is necessary after selecting a subset of the views, and (ii) SR-TA^{sel} and SR-NRA^{sel} become instance optimal.

Let $\mathbf{V}$ be the class of *sets $\mathcal{V}$ of pairwise disjoint views*, i.e., s.t. $Q^{V_i} \cap Q^{V_j} = \emptyset, \forall V_i, V_j \in \mathcal{V}, V_i \neq V_j$. We say an algorithm $A \in \boldsymbol{A}$ is *instance optimal over* $\boldsymbol{A}$ *and* $\boldsymbol{V}$ if $A \preceq B, \forall B \in \boldsymbol{A}$ and $\forall \mathcal{V} \in \boldsymbol{V}$. We can prove the following:

Theorem 1. SR-TA^{sel} *is instance optimal over* $\boldsymbol{A}$ *and* $\boldsymbol{V}$.
SR-NRA^{sel} *is instance optimal over* $\boldsymbol{A}$ *and* $\boldsymbol{V}$*, when only sequential accesses are allowed.*

Intuitively, for this class of views, the only way to obtain bounds for a query Q is the following: (i) for lower-bounds, only the views V that have $Q^V \subseteq Q$ are taken into account, while (ii) for upper-bounds all views V that verify $Q^V \cap Q \neq \emptyset$ are used. Note that this method is in effect the view selection algorithm for the class of pairwise disjoint views. Note also that the setting of [7], i.e. per-attribute lists of exact scores, is strictly subsumed by $\mathbf{V}$.

6 Context Transposition

We have discussed until now how queries can be answered by exploiting pre-computed results from views, with the important assumption that these share the same context with the input query. We remove now this restriction, and consider also views that may have been computed in a different context. We show how we can still answer input queries by the techniques discussed so far, by

pre-processing views in order to place them in the context of the input query. We call this step the *context transposition*.

Let us consider the two motivating applications of context-aware search: location-aware search and social-aware search. In both applications, one view V's context $\mathcal{C}^V$ can be seen as consisting of

1. a *location* (or *start point*) $\mathcal{C}^V.l$, e.g., geo-coordinates in a multidimensional space for location-aware search, or the social identity of a seeker in social-aware search,

2. a *contextual parameter* $\mathcal{C}^V.\alpha$, which basically parameterizes the influence of the spatial or social aspect in scores.

Given an input query Q, a context $\mathcal{C}$ – with $\mathcal{C}.l$ and $\mathcal{C}.\alpha$ – and a view V with a different context (either the location or α may differ, or both), in order to be able use pre-computed results from V, we need to derive from the existing $ans(V)$ tuples new score bounds: for each $(o, wsc, bsc) \in ans(V)$ we want to obtain a new tuple $(o, f_w(wsc), f_b(bsc))$. The functions f_w and f_b represent the core of the context transposition, their role being to map the worst scores and best scores of objects from $ans(V)$ to new guaranteed bounds for context $\mathcal{C}$.

7 Experiments

We performed our experiments on a single core of a i7-860 2.8GHz machine equipped with 8GB of RAM. We implemented our algorithms in Java, and we used this implementation for our tests on synthetic data and social data. We also implemented them in C++, for a more reliable comparison with IR-TREE, for spatial data.

Context-agnostic setting with complete views. Our first series of tests, over synthetic data, concerns a setting in which the input queries and the views share the same context (i.e., context plays no role and is ignored in the computation). We generated exact scores in the range $[0, 100]$ for 100,000 objects and 10 attributes, with exponential or uniform distributions. Then, we generated all possible combinations of 2 and 3 attributes, each representing one view. For each of the views, we computed the exact (aggregated) scores over *all* objects; the views are *complete* in that sense. We then made these lists uncertain by replacing each exact value by a score range, using the gaussian distribution with mean equal to the exact value and standard deviation (std, in short) equal to either 5, 10 or 20. Over the sets of views obtained in this way, we used 100 randomly-generated input queries consisting of 5 distinct attributes.

We compare in Figure 1 the SR-TA variants over the two data distributions, for the std values 5 and 10 (to avoid clutter, the plots for std 20 are not given). We have recorded (i) the relative running-time of the algorithms that use view selection w.r.t. the algorithm using all the views – three selection criteria per two std values, for six plot lines, (ii) the number of sequential accesses by all four variants – with the two std values, for eight plot lines, and (iii) the number of random accesses by all four variants – with the two std values, for eight plot lines.

One can note that the algorithms with view selection achieve significant savings in terms of both running-time and I/O accesses. The algorithm based on max-statistics, SR-TA^{max}, achieves

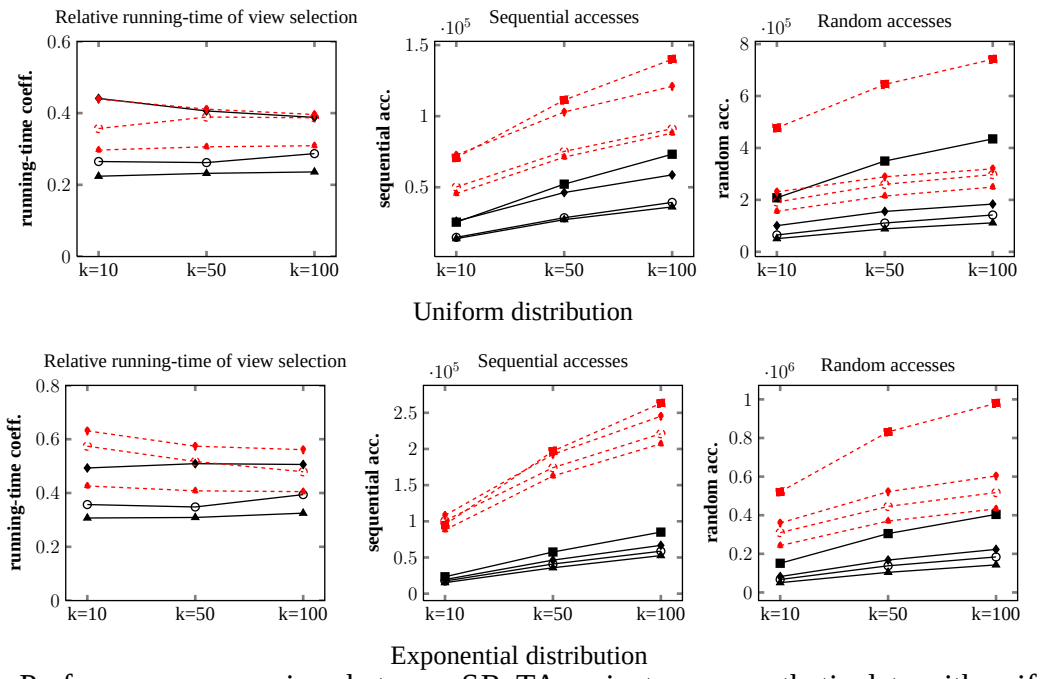

Figure 1: Performance comparison between SR-TA variants over synthetic data with uniform and exponential distribution.

Sel. + Dist.	Rel. running-time			Min. precision			$\|P\|$		
	10	50	100	10	50	100	10	50	100
`avg` + uni	0.576	0.676	0.712	0.57	0.69	0.72	10	36	64
`def` + uni	0.350	0.446	0.544	0.57	0.69	0.72	10	36	64
`max` + uni	0.296	0.395	0.446	0.57	0.69	0.72	10	36	64
`avg` + exp	0.732	1.128	1.287	0.60	0.63	0.64	10	46	86
`def` + exp	0.531	0.771	1.003	0.60	0.63	0.64	10	46	86
`max` + exp	0.456	0.684	0.827	0.60	0.63	0.64	10	46	86

Table 3: Comparison between SR-TA and TA (exact scores), for uniform and exponential distributions, for std 5.

better performance than the one based on view definitions, SR-TA^{def}, which in turn does better than the one based on average-statistics, SR-TA^{avg}. Furthermore, we can observe that the relative running-time of these algorithms does not depend on the value of k, and the influence of the interval coarseness (by standard deviation) is more important in the exponential distribution. One can also note a "clustering" effect, by standard deviation, in the case of sequential-access measures; this is likely due to the fact that top-k processing on noisier data needs to go deeper in the views to reach termination.

We also compared the performance of SR-TA, over score ranges with low noise (std of 5), with the one of Fagin's TA over the exact per-attribute inverted lists. We trace two measures: the relative running-time and the minimum precision. The latter is computed as $|G|/k$, i.e., the ratio between the size of the guaranteed set and the required k. The results are presented in Table 3. One can note that SR-TA^{sel} can have a running-time that is a low fraction of that of TA (as low as 0.296, with a

Sel.	Std	Overhead	$\|\mathbf{G}\| - \|\tilde{\mathbf{G}}\|$	$\|\mathbf{P}\| - \|\tilde{\mathbf{P}}\|$	$\mathbf{\Delta}$
avg	5	0.031	38	-208	1.96×10^{70}
avg	10	0.033	35	-734	4.14×10^{129}
avg	20	0.119	15	-4828	2.65×10^{212}
def	5	0.040	37	-206	2.76×10^{69}
def	10	0.038	34	-727	1.96×10^{129}
def	20	0.138	15	-4749	5.93×10^{211}
max	5	0.041	35	-179	5.54×10^{64}
max	10	0.041	33	-575	6.38×10^{119}
max	20	0.117	15	-3592	7.96×10^{200}

Table 4: Running-time overhead and Δ difference, for SR-TA^{sel} with final refinement versus SR-TA^{sel} without refinement , for k=100 and exponential distribution.

precision@10 of 0.577). This is mainly due to the fact that, although inexact, we have aggregated scores pertaining to 2 or 3 query terms, while the noise levels are rather low. While using exact lists of aggregated data for top-k processing would certainly improve efficiency, as shown in [11], our experiments show that even relatively noisy aggregated data can lead to improvements, with reasonable precision.

Finally, we give in Table 4 the overhead of the refinement step discussed in Section 4, which uses random-accessing to refine a result ($\tilde{G}$, $\tilde{P}$) to the most informative one, (G,P). Overhead is measured as the ratio between the running-time of the base algorithm and the one of the refined algorithm. We also report on the Δ measure. Note that, while the number of possible combinations that are "avoided" increases exponentially with the standard deviation, the overhead of additional I/O accesses is small (range 3%-13%).

Location-aware search. The dataset used in this setting is the PolyBot one, provided by the authors of [4]. It consists of 6,115,264 objects (documents) and their coordinates in a 2D space, and a total of 1,876 attributes (terms). We have generated 20 views defined by 2-term queries at 5 different locations, varying the size of their ans lists (500, 1000 and 2000 entries). We used 10 to-be-answered queries at 5 locations (different to the ones of views) and we varied $k \in \{10, 20\}$ and $\alpha \in \{0.7, 0.8, 0.9\}$. For the α values, we used values close to those indicated by the authors of [5].

The algorithm we use as baseline in our evaluation is our implementation of the IR-TREE of [5]. It is based on R-tree indices [8], whose nodes are enriched with inverted lists consisting of the documents located inside the rectangle defined by the node. The algorithm maintains a priority queue, containing either objects and their scores, or tree nodes and the maximum scores in their inverted list. The algorithm alternates between visiting nodes and adding objects to the candidate list. It stops when k objects have been retrieved. Our implementation of this algorithm achieves very similar running-time to the one reported in [5].

We present in Figure 2 the results for relative running-time and precision. The relative running-time is computed as the ratio between the running-time of SR-TA and the one of IR-TREE. Precision is computed as the percentage of top-k items returned by SR-TA that also appear in the output of IR-TREE. Here, we used the sampling method from Section 3 to obtain the most likely top-k from the (G,P) answer, through 1,000 rounds of uniform sampling.

One can note that, for high values of α and low values of k, the response time of SR-TA is significantly lower than that of the IR-TREE (in practice, of the order of milliseconds), with

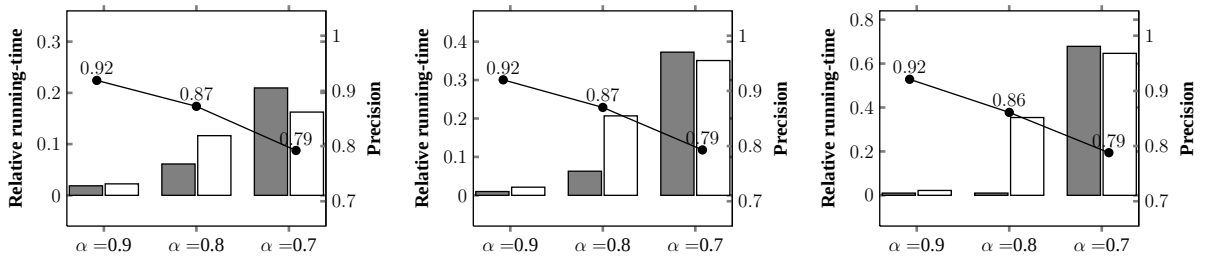

Figure 2: Location-aware search: performance and precision of SR-TA^{sel} versus exact early-termination algorithm (IR-TREE [5]), for various α values and list sizes (grey=top-10, white=top-20).

reasonably high precision levels (between 0.86 and 0.92). This is because the top-k answer is based on a large set G of guaranteed objects, which reduces the overhead of the sampling procedure. When the uncertainty introduced by coarser score ranges in views leads to larger sets P instead, the sampling procedure is more costly, but overall the running-time remains a small fraction of the one of the IR-TREE, with a precision around 0.8.

Social-aware search. For this application scenario, we used the publicly-available Delicious bookmarking data of [16]. We extracted a random subset of it, containing 80,000 users, their tagging behavior on 595,811 objects (items) with 198,080 attributes (tags). For assigning weights to links between users, we generated three similarity networks, by computing the Dice coefficients of either (i) common tags in a tag similarity network, (ii) common items in an item similarity network or (iii) common item-tag pairs in an item-tag similarity network.

For each of the three similarity networks, we randomly chose 5 seekers for our tests. Then, a number of 10 users were randomly chosen, among those having a link with weight of at most 0.66 to any of the 5 seekers (to ensure that no view is too "useful", having too strong an influence on the running-time and precision). For each of these users and for $\alpha \in \{0.0, 0.1, 0.2, 0.3\}$, we generated 40 views of 1 and 2-tag queries, each containing 500 entries.

The tests were made on a set of 10 3-tag queries for each of the 5 seekers, varying $\alpha \in \{0.0, 0.1, 0.2, 0.3\}$ and $k \in \{10, 20\}$.

The baseline algorithm we used for the performance comparison is a direct adaptation of the CONTEXTMERGE algorithm of [14]. In short, depending on the value of α, CONTEXTMERGE alternates between per-attribute inverted lists of objects and an inverted list containing users ordered descending by their proximity relative to the seeker. When the algorithm visits a user, her relevant objects – those that were tagged by her with attributes appearing in the input query – are retrieved and added to the candidate list. In manner similar to NRA, the algorithm keeps a threshold value representing the maximal possible score of objects, based on the maximal scores from the inverted lists and the proximity value of not yet visited users. The termination condition is very similar to that of NRA.

Similar to the location-aware search, we present in Figure 3 the results in terms of relative running-time and precision. One can note that the running-time is still a low fraction of the one of the exact algorithm, while the precision levels are considerably higher than in the case of location-aware search. As expected, the lowest precision levels are obtained when the search relies exclu-

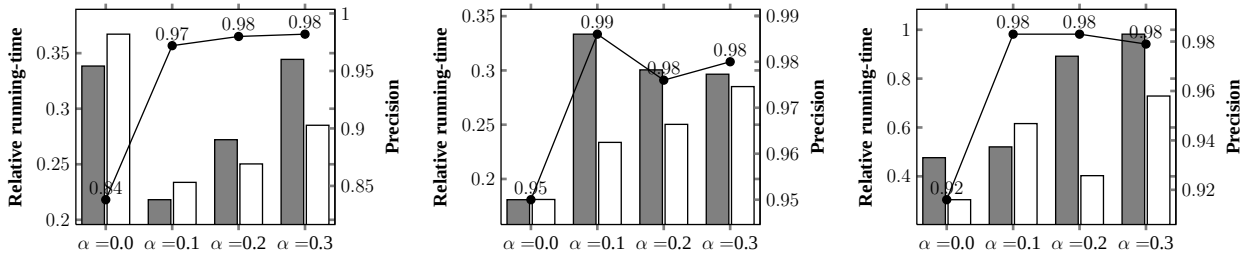

Figure 3: Social-aware search: performance and precision of SR-TA^{sel} versus exact early-termination algorithm (CONTEXTMERGE [14]), in three similarity networks (grey=top-10, white=top-20).

sively on the social component of the score. This is due to the fact that the bounds computed by the context transposition in social search yield coarser score ranges when $\alpha = 0$, which are source of more uncertainty in the scores and the top-k result. Moreover, due to the skew in proximity values in the network, even when α has low non-zero values, the textual component has a strong influence in scores, and thus leads to significant improvements in the top-k estimates (the most likely result).

8 Main Related Work

The most common data structure for top-k processing is the inverted index file (for a general survey on indexing for top-k processing see [19]), over which a key challenge is to optimize response time [17, 18]. Regarding algorithms, among the most widely cited and used are the early-termination threshold algorithms TA and NRA of [7], which provide instance optimality guarantees. Many other top-k aggregation algorithms have been proposed in the literature, and we refer the interested reader to the survey [10] and the references therein. The use of precomputed results, either as previous answers to queries [6, 9] or as cached intersection lists [11], has been identified as an important direction for efficiency. A linear programming formulation over score information is first introduced in [6] and extended in [11]. In [15], the authors study top-k processing when only score ranges are known, instead of exact ones, define a probabilistic ranking model based on partial orders and introduce several semantics for ranking queries, but do not deal with aggregation of uncertain scores over multiple dimensions. In the area of location-aware retrieval, Cong et al. [5] introduce the concept of LkT queries, for which they include in the ranking model both the distance of a document's location w.r.t. the query point and the textual features of the document. They propose the IR-tree index, consisting of an R-tree [8] in which each node has an inverted list of relevant documents. Other models for top-k location-aware keyword querying have been proposed, for selecting either groups of objects that collectively satisfy a query [2], or the k-best objects scored by the features in their neighborhood [13], or the top-k objects in a given query rectangle [4]. Various approaches for combining textual inverted lists and spatial indexes for keyword retrieval were also studied in [4]. In the area of social-aware search, for which bookmarking applications are a popular abstraction, processing top-k queries while having the social network as an integral part of the ranking model has been considered in recent research. [1] is the first to consider this problem, yet under significant restrictions, taking into account only a subset of

users and their documents in answers. The CONTEXTMERGE algorithm [14] is the first to address the social-aware search without imposing limitations on the exploration space, and they use the ranking model that we adopted in this paper. In [3], personalization based on a similarity network is shown to outperform other personalization approaches and non-personalized search.

9 Future Work

Our formulation of the view selection sub-problem opens many directions for future research. For example, finding combinatorial algorithms for view selection seems a promising approach for further reducing the overhead of LP computations. Regarding the refinement to the most probable result, it would be interesting to consider also approaches that can rely only on a subset of the remaining views, possibly in an incremental manner. The study of more refined definitions of classes of views that guarantee instance optimality is another avenue of further research.

With respect to applications, other context-aware scenarios, application-dependent context transpositions, as well as other ranking models, remain to be studied. Another important research direction is to study cost models that could help query processors in prioritizing view-based top-k computations over exact top-k computations.

References

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

[2] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, 2011.

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

[4] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel. Text vs. space: Efficient geo-search query processing. In *CIKM*, 2011.

[5] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. In *VLDB*, 2009.

[6] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis. Answering top-k queries using views. In *VLDB*, 2006.

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

[8] A. Guttman. R-tree: a dynamic index structure for spatial searching. In *SIGMOD*, 1984.

[9] V. Hristidis and Y. Papakonstantinou. Algorithms and applications for answering ranked queries using ranked views. In *VLDBJ*, 2004.

[10] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. In *ACM Comp. Surv.*, 2008.

[11] R. Kumar, K. Punera, T. Suel, and S. Vassilvitskii. Top-k aggregation using intersections of ranked inputs. In *WSDM*, 2009.

[12] S. Maniu and B. Cautis. Taagle: Efficient, personalized search in collaborative tagging networks. In *SIGMOD*, 2012.

[13] J. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Norvag. Efficient processing of top-k spatial preference queries. In *VLDB*, 2010.

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

[15] M. A. Soliman, I. F. Ilyas, and S. Ben-David. Supporting ranking queries on uncertain and incomplete data. In *VLDBJ*, 2010.

[16] R. Wetzker, C. Zimmermann, and C. Bauckhage. Analyzing social bookmarking items: A del.icio.us cookbook. In *ECAI Mining Social Data Workshop*, 2008.

[17] H. Yan, S. Ding, and T. Suel. Inverted index compression and query processing with optimized document ordering. In *WWW*, 2009.

[18] J. Zhang, X. Long, and T. Suel. Performance of compressed inverted list caching in search engines. In *WWW*, 2008.

[19] J. Zobel and A. Moffat. Inverted files for text search engines. In *ACM Comp. Surv.*, 2006.