Context-Aware Top-k Processing using Views

Extended Version

Silviu Maniu
Télécom ParisTech; CNRS LTCI
Paris, France
silviu.maniu@telecom-paristech.fr

Bogdan Cautis
Télécom ParisTech; CNRS LTCI
Paris, France
bogdan.cautis@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 difference in context between the various views and the input query leads to views containing objects 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.

1. INTRODUCTION

Retrieving the top-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 [11,15] or highly-efficient data structures [27] (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 [16,9].

At the same time, with the advent of location-aware devices, geotagging, 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 [9,20,13]. 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 score ranges from them 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 Versailles context, as one has to adapt the scores of restaurants from the pariscien perspective to the versaillais 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 [11] can support early-termination and output unranked results with only score ranges (instead of a precise ranking).

We give next two example scenarios – mostly self-explanatory – from location-aware and social-aware search. They illustrate, on one hand, the fact that previous (cached) results pertaining to one context may be interpretable only as uncertain, by score intervals, when dealing with a new query and a new context. On the other hand, they illustrate the fact that it may be possible to corroborate such uncertain descriptions (of scores of objects) from different views, in order to build a most refined or informative approximation of the top-k result that would be obtained by looking at the actual data instead of the views.

Motivation 1: Location-aware search. Let us consider the spatial-search scenario in Figure 1 in which we have objects at various locations in an euclidian space (objects $o_1, \ldots, o_5$ in the figure, as gray dots). Each object (e.g., a Web document) is characterized by a bag of attributes. For instance, $o_5$ has attributes $t_1$ and $t_2$, both with a single occurrence.

Now, users located at various points request the top-k objects
with respect to a set of attributes. In response, objects are ranked by a combination between the distance of the object w.r.t. the seeker’s location and the object’s content. While the details of the spatial ranking model will be clarified in Section \( \text{IV} \), let us assume in the following that the location of an object contributes 30\% to the score of an object. The remaining 70\% represent the weight of the textual score (e.g., tf-idf measures).

Consider a new query \( q \) in the system, asking for the top-2 items for attributes \( \{t_1, t_2\} \) at the point marked by a white dot in the figure. Intuitively, spatial search algorithms [6], by using indices such as the R-tree [12], would proceed by incrementally increasing the search distance until enough objects are found. However, an alternative execution plan may be possible, if we assume access to cached results of previous queries (initiated at the black dots).

For example, let us assume that \( V_1 \) gives the top-3 documents for \( \{t_1, t_2\} \), as the ranked list \([o_5 = 1.062, o_4 = 1.029, o_2 = 1] \). Also, sharing the same location, we have \( V_2 \) and \( V_3 \). The former gives the top-4 for \( \{t_1\} \) as \([o_2 = 0.946, o_3 = 0.575, o_5 = 0.425, o_4 = 0.262] \). The latter gives the top-4 for \( \{t_2\} \) as \([o_4 = 0.962, o_1 = 0.437, o_5 = 0.425, o_2 = 0.216] \).

Since \( V_1, V_2 \) and \( V_3 \) are closer to \( Q \) than any of the objects, it would be tempting to use their lists of pre-computed results, instead of looking for the actual objects.

In particular, one may resort to using only the results of \( V_1 \), as it is the closest to \( Q \) both spatially and textually. For that, we need first to perform a change of context, to account for the fact that objects that were close to \( V_1 \) may be even closer to \( Q \), as they may be farther. This will introduce uncertainty in the scores of \( V_1 \)’s result set. More precisely, knowing that the normalized distance between \( Q \) and \( V_1 \) is 0.175, for \( Q \)’s perspective, \( V_1 \)’s list should now have objects with scores interval [0, 1.062 − 0.3 × 0.35, 1.062 + 0.3 × 0.35], \([o_4 = 1.029 − 0.3 × 0.35, 1.029 + 0.3 × 0.35], o_2 = [1 − 0.3 × 0.35, 1 + 0.3 × 0.35] = \{o_5 ∈ [0.957, 1.167], o_4 ∈ [0.924, 1.134], o_2 ∈ [0.895, 1.105]\} \[10\].

We can see that \( V_1 \)’s result is not sufficient to answer \( Q \) with certainty, since any object among the three candidates may be in the top-2. Yet the solution can come from \( V_2 \) and \( V_3 \), albeit more distant, if we corroborate their results with the ones of \( V_1 \). Knowing that \( V_2 \) and \( V_3 \) are at a normalized distance of 0.25 the rewritten result sets would be, for \( V_1 \) \([o_2 = 0.871, o_3 ∈ [0.35, 0.5], o_4 ∈ [0.187, 0.337]\) and for \( V_2 \) \([o_4 ∈ [0.887, 1.037], o_5 ∈ [0.362, 0.512], o_3 ∈ [0.35, 0.5], o_2 ∈ [0.171, 0.321]\) \[10\].

We can see that \( V_1 \) result is not sufficient to answer \( Q \) with certainty, since any object among the three candidates may make it in the top-2. Yet the solution can come from \( V_2 \) and \( V_3 \), albeit more distant, if we corroborate their results with the ones of \( V_1 \). Knowing that \( V_2 \) and \( V_3 \) are at a normalized distance of 0.25 the rewritten result sets would be, for \( V_1 \) \([o_2 ∈ [0.871, 0.35] \), is obtained as 0.175 + 0.175, since the query has two tags.\[10\]
It can hence be seen that, after visiting just the neighbors $v1$ and $v2$, the search for $s$’s query can give the top-3 objects as $\{o1 \in [3.07, 3.8], o5 \in [2.27, 2.81], o2 \in [1.21, 1.5]\}$. Obviously, with fewer views, we may no longer reach the most informative answer (as presented a trade-off between the size of the selected subset of views and the size of cached results, in the form of views over the data).

The general goal of our work is to enable efficient context-aware top-$k$ retrieval that exploits exclusively the views. The rationale for this 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 uncertain pre-computed 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. Hence, 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 some 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, we may no longer reach the most informative answer $(G, P)$, and we are in general presented a trade-off between the size of the selected subset of 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.

On top of 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.

Importantly, our algorithms provide a one-size-fits-all solution for many search applications that are context-dependent, and we show how they can be directly applied in our two motivating applications scenarios for context-aware search. For both scenarios, we also describe how the necessary step of context transposition, transforming scores or ranges thereof, valid in one initial context, to ranges that are valid in a new context, for ranking models combining textual and location dimensions.

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 area of applications.

**Outline.** The paper is organized as follows. We formalize the context-aware search setting and the problems being studied in Section 2. We present the adaptation of the TA and NRA algorithms in Section 3. A sampling-based method for obtaining the most likely top-$k$ is presented in Section 4. Our optimization approaches by view selection are formulated in Section 5. We study the formal properties of SR-TA and SR-NRA in Section 6. In Section 7, we present context transposition methods for location-aware and social-aware queries, and we give the outline for a generic context-aware query processor in Section 8. We present our experimental evaluation in Section 9. An overview of the related work is given in Section 10. Finally, we present our conclusions and we discuss future research in Section 11.

## 2. FORMAL SETTING AND PROBLEMS

**Context-aware score model.** We assume a finite collection of objects $O$ and a countable collection of attributes $T$. Under a given context parameter $C$ – an application-dependent notion – objects $o$ are associated to certain attributes $t$, by an object-attribute score function $sc(o, t \mid C)$.

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

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

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

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

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 now 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

1. an upper-bound on $sc(o, Q)$, the best possible score that may still be obtained for $o$, denoted hereafter $\text{bsc}(o, Q)$,
2. a lower-bound on $sc(o, Q)$, the worst possible score, denoted hereafter $\text{wsc}(o, Q)$,
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 $k$th object in $D$, $\text{wsc}(D[k], Q)$, with the best possible score of either (i) objects $o$ in $D$ outside the top $k$, $\text{bsc}(o, Q)$, or (ii) not yet encountered objects, denoted $\text{bsc}(s, Q)$. 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 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 these lists, as soon as an object $o$ has been encountered while sequentially accessing one list among the $(Q)$ relevant ones. These random accesses will 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-$k$ will only be given a score range, $[\text{wsc}(o, Q), \text{bsc}(o, Q)]$. (Various hybrid algorithms with respect to TA and NRA are also possible and have been extensively studied in the literature.)

TA was shown to be \textit{instance optimal} 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.$^4$

\section*{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 \textit{views}. Each view $V$ has two components: (i) a definition, $\text{def}(V)$, which is a pair query-context $\text{def}(V) = (Q^V, \mathcal{C}^V)$ and (ii) a set $\text{ans}(V)$ of triples $(o_i, \text{wsc}_i, \text{bsc}_i)$, representing the \textit{answer} to query $Q^V$ under context $\mathcal{C}^V$. Each triple says that object $o_i$ has a score $\text{sc}(o_i, Q^V | \mathcal{C}^V)$ within the range $[\text{wsc}_i, \text{bsc}_i]$.

Since we are dealing with cached query results, all objects not appearing in $\text{ans}(V)$ – represented explicitly in $\text{ans}(V)$, to simplify presentation, by one final \textit{wildcard object} – have with respect to query $Q^V$ and context $\mathcal{C}^V$ a worst score of $\text{wsc}_i = 0$ and a best possible score of either

1. $\text{bsc}_i = \min \{\text{wsc}_i, \text{bsc}_i \in \text{ans}(V)\}$, if $V$’s result is complete, in the sense that enough objects had a non-zero score w.r.t. $Q^V$,  
2. 0, otherwise.

\section*{Context transposition.} Intuitively, when a view $V$ and the to-be-answered query $Q$ do not share the same context, a transposition of the score ranges in $\text{ans}(V)$ is necessary, in order to obtain valid ranges for $\text{sc}(o_i, Q^V | \mathcal{C})$ from those for $\text{sc}(o_i, Q^V | \mathcal{C}^V)$. 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 4.

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

\section*{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 $(\text{sc}, h)$ and a set of views $V$ sharing the same context with $Q$, given an object $o \in \mathcal{O}$, compute the tightest lower and upper bounds on $\text{sc}(o, Q | \mathcal{C})$ from the information in $V$.

In this paper, consistent with the most used 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:

\begin{align}
\min_{t_i \in Q} & \sum_{t_i \in Q} \text{sc}(o, t_i \mid \mathcal{C}) \\
\max_{t_i \in Q} & \sum_{t_i \in Q} \text{sc}(o, t_i \mid \mathcal{C}) \\
\text{wsc} \leq & \sum_{t_i \in Q^V} \text{sc}(o, t_i \mid \mathcal{C}), \forall V \in V \text{ s.t. } (o, \text{wsc}, \text{bsc}) \in \text{ans}(V) \\
\text{sc}(o, t_j \mid \mathcal{C}) \leq & \text{bsc}, \forall V \in V \text{ s.t. } (o, \text{wsc}, \text{bsc}) \in \text{ans}(V) \\
\text{sc}(o, t_i \mid \mathcal{C}) & \geq 0, \forall t_i \in \mathcal{T}
\end{align}

whose variables are given in bold.

\section*{Example 1.} Let us consider the views in Table 4. We have access to the results of four views, defined by $\{a\}$, $\{c\}$, $\{a, b\}$ and $\{b, c\}$. We assume the empty context $\mathcal{C} = \bot$ for the views and for the to-be-answered query, which is $Q = \{a, b, c\}$. Considering $\text{sc}$, for example, we know that:

\begin{align}
\text{sc}(\text{sc}(\{a\}), 4) & \geq 4 \\
\text{sc}(\text{sc}(\{c\}), 2) & \geq 2 \\
\text{sc}(\text{sc}(\{a\}) + \text{sc}(\{c\}), 10) & \geq 10 \\
\text{sc}(\text{sc}(\{a\}) + \text{sc}(\{c\}), 9) & \geq 9 \\
\text{sc}(\text{sc}(\{a\}) + \text{sc}(\{c\}), 4) & \leq 4 \\
\text{sc}(\text{sc}(\{c\}), 4) & \leq 4 \\
\text{sc}(\text{sc}(\{a\}) + \text{sc}(\{b\}), 10) & \leq 10 \\
\text{sc}(\text{sc}(\{b\}) + \text{sc}(\{c\}), 11) & \leq 11
\end{align}

Then, the lower bound on $\text{sc}(\{a\}, Q)$ is obtained as

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

by combining the worst scores of $V_1$ and $V_4$. Similarly, the upper bound on $\text{sc}(\{a\}, Q)$ is obtained as

$\text{bsc}(\{a\}) = \max(\text{sc}(\{a\}) + \text{sc}(\{b\}) + \text{sc}(\{c\})) = 14$

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 \textit{most informative result} would consist of two disjunctive, possibly-empty sets of objects from those appearing in $V$:

\begin{itemize}
\item a set of all the objects guaranteed to be in the top-$k$ for $Q$  
\item a set of all objects that may also be in the top-$k$ for $Q$.
\end{itemize}
Table 1: An example set $V$ of views.

<table>
<thead>
<tr>
<th>$V_1({a}, \perp)$</th>
<th>$V_2({c}, \perp)$</th>
<th>$V_3({a, b}, \perp)$</th>
<th>$V_4({b, c}, \perp)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>o</td>
<td>ws</td>
<td>bs</td>
<td>o</td>
</tr>
<tr>
<td>o3</td>
<td>7</td>
<td>8</td>
<td>o3</td>
</tr>
<tr>
<td>o5</td>
<td>6</td>
<td>7</td>
<td>o4</td>
</tr>
<tr>
<td>o6</td>
<td>4</td>
<td>4</td>
<td>o6</td>
</tr>
<tr>
<td>o7</td>
<td>3</td>
<td>5</td>
<td>o10</td>
</tr>
<tr>
<td>o9</td>
<td>3</td>
<td>4</td>
<td>o7</td>
</tr>
<tr>
<td>o10</td>
<td>1</td>
<td>3</td>
<td>o8</td>
</tr>
<tr>
<td>o2</td>
<td>1</td>
<td>2</td>
<td>o9</td>
</tr>
<tr>
<td>o1</td>
<td>1</td>
<td>1</td>
<td>o5</td>
</tr>
<tr>
<td>*</td>
<td>0</td>
<td>1</td>
<td>*</td>
</tr>
</tbody>
</table>

The former kind – the guaranteed ones – as the objects $o_a$ for which

$$\min \sum_{t_i \in Q} sc(o_a, t_i | C) \geq \max \sum_{t_i \in Q} sc(\ast, t_i | C) \quad (2.3)$$

and at most $k−1$ objects $o_a$ can be found such that

$$\min \sum_{t_i \in Q} sc(o_a, t_i | C) < \max \sum_{t_i \in Q} sc(o_y, t_i | C). \quad (2.4)$$

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

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

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

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

- $G \subseteq Q$ consisting of all guaranteed objects (as in Eq. (2.3)) and (2.4): they must be among those with the $k$ highest scores for $Q$ and $C$,

- and $P \subseteq Q$ consisting of all possible objects outside $G$ (as in Eq. (2.3)): they may be among those with the highest scores for $Q$ and $C$, i.e., there exist data instances where these appear in the top-$k$.

In order to solve Problem 3, a naive 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 5 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 5 possible approaches for estimating the probability of possible top-$k'$ sets from $P$.

**View selection.** 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 4). This is because the set of views may be too large – potentially of the order $2^{\lceil T \rceil}$ – and each view contributes one constraint in the program.

We consider a best-effort approach, which would first select some (few) most promising views $\tilde{V} \subseteq V$ for the input query, before running the threshold algorithm. In doing so, we are presented a trade-off between the size of the subset $\tilde{V}$ – 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 $\tilde{V}$, $(\tilde{G}, \tilde{P})$, and the most informative answer $(G, P)$ by $V$ as the difference in the number of possible top-$k$ combinations:

$$\Delta = \left( \frac{\tilde{P}}{k - |\tilde{G}|} \right) - \left( \frac{|P|}{k - |G|} \right). \quad (2.6)$$

We present in Section 5 our approach for view selection. We also show how a final refinement step over $(\tilde{G}, \tilde{P})$, based on random accesses in the entire $V$ set, allows us to reach $\Delta = 0$, i.e., the most informative result by $V$.

**Main research landscape.** We have already introduced TA and NRA, as reference algorithms for early-termination top-$k$ retrieval. Techniques for top-$k$ answering using views have been proposed in recent literature: the LPTA algorithm [9] and generalizations of the NRA and TA algorithms [10], both applicable in settings where the aggregation functions are linear combinations of the per-attribute scores. These two approaches make however significant simplifying assumptions: (i) the scores in each view are assumed exact, i.e., $w_{sc} = b_{sc}$, $\forall (a, w_{sc}, b_{sc}) \in ans(V)$, $\forall V \in V$, and (ii) one can compute the score for the input query $Q$ by composing (in a predefined way) the scores from a subset of the views that are selected in advance (usually the single-attribute inverted lists). In comparison, we consider more general views, that may only give score ranges instead of exact scores and ranks in the top-$k$ result. In our more general setting, the exact top-$k$ may not be obtainable will full confidence and, instead of simply refuting input queries that cannot be fully answered, we describe algorithms that can support a more general type of result from the views, in terms of guaranteed and possible objects for the top-$k$.

### 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 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 $\tau$ – cannot be greater than the one of the $k$th object in the candidate set $D$.
view $V$ the score upper-bound of objects from $\text{ans}(V) - D$:

$$\tau = \max \sum_{t_j \in Q} \text{sc}(o, t_j \mid C)$$ (3.1)

$$\sum_{t_j \in Q^V} \text{sc}(o, t_j \mid C) \leq \max(\text{bsc}_i), \forall V \in \mathcal{V}, o_i \notin D \text{ s.t.}$$

$$(o_i, \text{wsc}_i, \text{bsc}_i) \in \text{ans}(V)$$

Remark. One can note that when (i) we have views that only contain answers to singleton queries, and (ii) the $\text{wsc}_i = \text{bsc}_i$ 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 [10][9]. 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.

**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 item $o1$, or 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 candidate 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 flow is similar to the one of TA, with the notable addition of the generalized computation of upper and lower bounds and of the threshold value.

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

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

$$0 \leq \sum_{t_j \in Q^V} \text{sc}(o, t_j \mid C) \leq \max\{\text{bsc}_i \mid (o_i, \text{wsc}_i, \text{bsc}_i) \in V, o_i \notin D\}$$ (3.2)
The termination conditions need to keep track, besides the threshold value, of the upper maximum-bound of partially known objects that are in the current top-k of D, denoted $bsc_{pre}$. Objects that are fully known are ignored in this estimate, since their scores are fully filled and they might be candidates for $P$. The general flow of SR-NRA is given in Algorithm 3.

### Partition for most informative result

Once the main loop of SR-TA or SR-NRA terminates, the 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 4 details this step: for each object $o$ in $D$ we simply test the conditions of Eq. (2.1), (2.3), and (2.4).

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 $o \in O$ is in the set $G$ outputted by \textsc{Partition}$\langle D, k \rangle$ if and only if in all possible data instances $o$ is the top-k for $Q$ and $C$.

An object $o \in O$ is in the set $P$ outputted by \textsc{Partition}$\langle D, k \rangle$ if and only if in at least one possible data instance $o$ is in the top-k for $Q$ and $C$.

#### Algorithm 3: \textsc{Partition}$\langle D, k \rangle$

**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 $(o, bsc, wsc) \in D$, $o \neq \ast$ do
4: $x \leftarrow \{(o', bsc', wsc') \in D \mid o' \neq o, bsc' > wsc\}$
5: $wsc \leftarrow$ lower-bound score of kth candidate in $D$
6: if $t < k$ and for $(*, wsc_*, bsc_*) \in D, bsc_\leq wsc$ then
7: add $o$ to $G$
8: else if $bsc > wsc_\leq$ then
9: add $o$ to $P$
10: end if
11: end for
12: return $G, P$

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

**Remark.** We have only detailed here the adaptation of TA and NRA, the reference algorithms for early-termination top-k retrieval. However, more generally, any early-termination algorithm, as the ones following a hybrid approach between the two, can be adapted in similar manner to the setting with cached results having score ranges.

### 4. Extracting a Probable Top-k’ in $P$

As described in Section 2, 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, based on $G$ and $P$, could be identified. Since for each object in $P$ we have a lower-bound and an upper-bound on its exact score, let us assume in the following a uniform probability-density function 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 $Pr(o_1 > o_2)$.

**Example 3.** Returning to Example 2 recall $P = \{\{0, 7, 8\}, \{0, 3, 9\}\}$, and $G$ consists of 4 objects. To complete the requested top-5 answer, we need to estimate the top object from $P$, the one that is most likely to have the highest score. Denoting this top object $t_1$, we can see that $Pr[t_1 = \{0, 7\}] = Pr[0T > 04]$ and $Pr[t_1 = \{04\}] = Pr[04 > 07]$. If we assume that each score is uniformly distributed within its known [wsc, bsc] interval, we have that $Pr[0T > 04] = 0.833$ and $Pr[04 > 07] = 0.166$.

Overall, the most likely top-5 answer to the input query is thus $\{03, 05, 06, 010, 07\}$, with a probability of 0.833.

A much more efficient algorithm than naïve enumeration is to adapt our setting the sampling-based approach of [21], which computes top-k answers over uncertain data, namely ranked object list with score ranges and probability-density functions over them.

We describe in Algorithm 4 a tractable approach for estimating the most likely top-k’ over a set of triples $(o, u,wsc, bsc)_i$, under the assumption that sampling can be done in polynomial-time as well. We use an encoding-decoding pair of functions that map sets of objects to numerical keys, and vice-versa: $key = \text{encode}(S)$ is the key representing the set $S$, and $S = \text{decode}(key)$ gives the opposite mapping.

We proceed in Algorithm 4 as follows. We first initialize a hash table $T$ for the domain of keys (range of $\text{encode}$). For a given number of sampling rounds, at each round $i$ we go through the objects from $P$ and generate for each a score based on its range; we then order the objects based on these scores into a list $P_i$ (sample_scores subroutine). We obtain through $\text{encode}$ the key for the set consisting of the top-k’ objects in $P_i$, 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 $T$.

#### Algorithm 4: \textsc{Estimate}$\langle P, k', r \rangle$

**Require:** objects $P$, parameter $k'$, $r$ sampling rounds

1: initialize hash table $T$
2: for rounds $l = \{1, \ldots, r\}$ do
3: $P_i \leftarrow \text{sample_scores}(P)$
4: $key = \text{encode}((P_i[1], \ldots, P_i[k']))$
5: if $key \notin T$ then
6: $T[\text{key}] = 0$
7: end if
8: $T[\text{key}] = T[\text{key}] + 1$
9: end for
10: $\text{key} = \text{argmax}(T[i])$
11: return $\text{decode}(\text{key})$

### 5. 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 than the $(G, P)$ one. We also 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 5.1 could be modeled straightforwardly by the mathematical programs 5.1 and 5.2. Put otherwise, we have as the dual of the minimization problem 5.1 the following packing LP:

\[
\max \sum_{i=1}^{\left| V \right|} w_{sc} \times l_{i} \quad s.t. \quad \sum_{t \in Q_{t}^{l}} l_{j} \leq 1, \forall t \in Q \quad (5.1)
\]

and we have as the dual the maximization problem 5.2 the following covering LP:

\[
\min \sum_{i=1}^{\left| V \right|} b_{sc} \times u_{i} \quad s.t. \quad \sum_{t \in Q_{t}^{l}} u_{j} \geq 1, \forall t \in Q \quad (5.2)
\]

Based on the programs 5.1 and 5.2 for each object \( o \), in order to obtain its most refined bounds, we would need to first fractionally select views from \( V \) – as opposed to integer choices – 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 \( V_{i} \in V \) such that (i) \( l_{i} \neq 0 \), for worst scores, or (ii) \( u_{i} \neq 0 \), for best scores.5

**Example 4.** Let us consider the views in Table 1, using the LPs 5.1 and 5.2 to illustrate view selection for object \( o6 \).

For the worst score, we need to optimize

\[
\max 4l_{1} + 2l_{2} + 10l_{3} + 9l_{4}, \quad s.t. \quad l_{1} + l_{3} \leq 1, \quad l_{3} + l_{4} \leq 1, \quad l_{2} + l_{4} \leq 1.
\]

The optimal is reached when \( l_{1} = 1, l_{2} = 0, l_{3} = 0 \) and \( l_{4} = 1 \), i.e., using the worst scores of \( o6 \) from views \( V_{1} \) and \( V_{4} \).

For the best score, we need to optimize

\[
\min 4l_{1} + 2l_{2} + 10l_{3} + 11l_{4}, \quad s.t. \quad l_{1} + l_{3} \geq 1, \quad l_{3} + l_{4} \geq 1, \quad l_{2} + l_{4} \geq 1.
\]

The optimal is reached when \( l_{1} = 0, l_{2} = 1, l_{3} = 1 \) and \( l_{4} = 0 \), i.e., using the best scores of \( o6 \) from views \( V_{2} \) and \( V_{3} \).

**Best-effort view selection.** Solving the LPs 5.1 and 5.2 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 \( w_{sc} \) and \( b_{sc} \) values, in an approximate version of the two LPs, each view \( V_{i} \)

5Restricting the domain of the \( u \) and \( l \) values to integers would lead to an NP-hard view selection problem. More precisely, Eq. 5.2 would reduce to an instance of the weighted set cover problem, and Eq. 5.1 would reduce to an instance of the \( k \)-dimensional perfect matching problem (where \( k = \max(|Q(V')|), \forall V \in V \)). 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.

could be represented by two unique values, \( w_{sc}(V) \) and \( b_{sc}(V) \). Our optimization problems would then simplify as follows:

\[
\max \sum_{i=1}^{\left| V \right|} w_{sc}(V_{i}) \times l_{i} \quad s.t. \quad \sum_{t \in Q_{t}^{l}} l_{j} \leq 1, \forall t \in Q \quad (5.3)
\]

\[
\min \sum_{i=1}^{\left| V \right|} b_{sc}(V_{i}) \times u_{i} \quad s.t. \quad \sum_{t \in Q_{t}^{l}} u_{j} \geq 1, \forall t \in Q \quad (5.4)
\]

and this would enable us to select the “good” views in the initialization phase of the top-\( k \) algorithm, as those that participate 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. 5.1 and 5.2 (lines 5-6 in SR-TA) by the following estimates that use only the selected views \( V' \):

\[
\hat{w}_{sc} = \sum_{i=1}^{\left| V' \right|} w_{sc} \times l_{i} \quad \hat{b}_{sc} = \sum_{i=1}^{\left| V' \right|} b_{sc} \times u_{i}
\]

This is possible since, by the duality property, we are guaranteed that the feasible solutions for Eq. 5.3 and 5.4 represent safe bounds for \( o \)'s scores, i.e., \( \hat{w}_{sc} \leq w_{sc} \) and \( \hat{b}_{sc} \geq b_{sc} \). We can similarly simplify Eq. 5.1, for the threshold value (for line 8 in SR-TA).

**Candidates for \( w_{sc}(V) \) and \( b_{sc}(V) \).** We followed the approach described above, which approximates view selection, in two distinct flavors.

First, the per-view score bounds \( w_{sc}(V) \) and \( b_{sc}(V) \) could be based solely on the view’s definition \( Q(V) \), and we experimented in this paper with bounds that are simply defined as

\[
w_{sc}(V) = b_{sc}(V) = |Q(V)|
\]

for each \( V \in V \). 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 2.2 natural per-view measures that are based on the views’ answers: (i) the average value of scores, and (ii) the maximum value of scores.

**Using the attribute-query graph.** An important optimization that can always be performed before the actual view selection is to filter out all views that cannot influence the score of objects w.r.t. the input query. For that, we can build an undirected graph of queries, in which we have (i) a node for the input query \( Q \), and (ii) one node for each of the views in \( V \). Two nodes are connected by an edge if they have at least one attribute in common. It is then easy to see that we can filter out all the views that are not in the connected component of the input query, as their objects’ scores will be useless for computing an answer to \( Q \).

**Retrieving \((G, P)\) after view selection.** We conclude this section by discussing how the most informative result \((G, P)\) – that can be obtained from the complete set of views \( V \) – can still be retrieved by refining a result \((G, P)\) obtained on a selection of views \( V \). 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 \( P \) by random-accessing their scores in all
the views in $V$.

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 $A = 0$. So 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 $V$. We describe in Section 9 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 is to replace the $sel$ superscript by a def, max or aeg one, depending on the selection method being used.

6. FORMAL GUARANTEES

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

Let $A$ 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 $V$, we denote by $D(V)$ the class of all instances of answers in those views, i.e., $ans(V)$, $V \in V$.

Given two algorithms $A_1 \in A$ and $A_2 \in A$, we write $A_1 \prec A_2$ iff, for all sets of views $V$ and all instances of the data $D$, $A_2$ 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). Conversely, we write $A_1 \not\prec A_2$ iff there exists at least one instance of the views $V$ and their data for which $A_2$ costs less than $A_1$. We say that an algorithm $A \in A$ is instance optimal over $A$ iff $A \not\prec B, \forall B \in A$.

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 sets of views $V$. The answer to this question is not at all obvious: on one hand, SR-TA$_{nosel}$ or SR-NRA$_{nosel}$ should use less views to compute the $P$ and $G$ sets, but they might either go too deep in the views or might need additional accesses in the other views, compared to SR-TA$_{sel}$ or SR-NRA$_{sel}$. On the other hand, SR-TA$_{sel}$ or SR-NRA$_{sel}$ might go through views that are useless for deriving optimal bounds. We can prove the following:

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

The proof is given in Appendix A.

**Lemma 1** tells us that neither of the two variants of SR-TA or SR-NRA can be instance optimal for all possible sets $V$. 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 $B$ be the class of sets of pairwise disjoint views, i.e., the class of all sets of views $V$ s.t. $Q^{i_1} \cap Q^{i_2} = \emptyset, \forall V_{i_1}, V_{i_2} \in V, V_{i_1} \neq V_{i_2}$. We say an algorithm $A \in A$ is instance optimal over $A$ and $V$ if $A \not\prec B, \forall B \in A$ and $\forall V \in V$. We can prove the following:

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

Intuitively, for this class of views, the only way to obtain bounds for any 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 $\{1, \}$, i.e. per-attribute disjoint views, is strictly subsumed in $V$.

7. 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 preprocessing views in order to place them in the context of the input query. We call this step the context transposition.

We give in this section the details on context transformations for our two motivating application scenarios: location-aware search and social-aware search. In both applications, one view $V$’s context $C^V$ can be seen as consisting of:

1. a location (or start point) $C^V.l$, e.g., geo-coordinates in a multidimensional space for location-aware search, or the social identity of a seeker in network-aware search.
2. a contextual parameter $C^V.\alpha$, which basically parameterizes the influence of the spatial or social aspect in scores.

Given an input query $Q$, a context $C$ – with $C.l$ and $C.\alpha$ – and a view $V$ with a different context (either the location or $\alpha$ 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 $(o, w, b) \in ans(V)$ we want to obtain a new tuple $(o, f\_w(se), f_b(bs))$. 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 bounds, which will be valid in the context $C$. We detail them next for each of the two application scenarios.

7.1 Location-aware search

In location-aware or spatial top-k querying, a user having a certain location is interested in the top-$k$ objects that are relevant textually and close spatially.

We revisit here one of the most common ranking models, in which the per-attribute score of an object is a linear combination of spatial relevance and textual relevance. Each object $o$ consists of a bag of attributes $o.A$ and a location $o.l$. Given an input query $Q$, with context $C$ having location and $C.l$ and parameter $C.\alpha$, the per-attribute score is obtained as follows:

$$sc(o, t | C) = (1 - C.\alpha)(1 - D(C.l, o.l))_{\max Dist} + C.\alpha TF(t, o.A)$$

(7.1)

where $D$ gives the euclidean distance between $Q$’s location (start point) and $o$’s location, $\max Dist$ is the maximal distance, $TF(t, o.A)$ is the term frequency of $t$ in $o.A$, and $maxTF(t)$ is a maximal term frequency of $t$ over all objects.

We now detail how context transposition is performed. For ease of exposition, we first describe how the location component of the context is transposed. Then, we describe the transposition for $\alpha$.

Recall that we are in a setting where we are presented only with partial information, in each $V$, in the form of tuples $(o, w, b)$, and we have access neither to $D(C.l, o.l)$ nor to $TF(t, o.A)$. The bounds might be tight, i.e., $w = b$, as in [3, 1], or may be loose.

**Transposing the location.** Given a query $Q$ of context $C$, with location $C.l$, given of view $V$ of context $C^V$, with location $C^{V.l}$, for
any object \( o \in \text{ans}(V) \), there are two extreme locations at which \( o \) can be situated, relative to \( C.I \) (see Figure 3), as follows:

1. on the segment between \( C.I \) and \( C.V.I \), or on the line connecting \( C.I \) and \( C.V.I \), beyond \( C.I \), resulting into
   \[
   D(C.I, o.I) = |D(C.V.I, C.I) - D(C.V.I, o.I)|
   \]
2. on the line connecting \( C.I \) and \( C.V.I \), beyond \( C.V.I \), giving
   \[
   \]

We can now derive the following new bounds for each object \( o \) from a tuple \( (o, \text{wsc}, \text{bsc}) \in \text{ans}(V) \), which would be valid in a context \( C' \) defined by the query’s location \( C.I \) and the view’s \( C.V.o \):

- \( \text{wsc} \) or \( \text{bsc} \):
  \[
  \text{wsc} \leq \text{wsc} - C.V.o \times |Q| \times \frac{D(C.V.I, C.I) \times \text{maxDist}}{\text{maxDist}} = \text{wsc}'
  \]
  \[
  \text{bsc} \leq \text{bsc} + C.V.o \times |Q| \times \frac{D(C.V.I, C.I) \times \text{maxDist}}{\text{maxDist}} = \text{bsc}'.
  \]

Transposing the parameter \( o \). We detail now the transposition for the component, from \( C.V.o \) to \( C.o \), by which are obtained bounds that are valid for the input query context \( C.I \).

Let in Eq. (7.1)

\[
x = 1 - \frac{D(C.I, o.I)}{\text{maxDist}}, \quad y = \frac{TF(t.o.A)}{\text{maxTF}(t)}.
\]

From the intermediary context \( C' \) introduced by the transposition of location, for any tag \( t \in Q^V \) and object \( o \in \text{ans}(V) \), we have:

- \( \text{wsc} \) or \( \text{bsc} \):
  \[
  \text{wsc} = (1 - C.V.o) \times x + C.V.o \times y
  \]
  \[
  \text{bsc} = (1 - C.o) \times x + C.o \times y.
  \]

This leads to

\[
\text{sc}(o, t | C) = \text{sc}(o, t | C') - (C.o - C.V.o) \times (x - y)
\]

Since \( x \in [0, 1] \) and \( y \in [0, 1] \), we have that

\[
\max(x - y) = 1, \quad \min(x - y) = -1,
\]

therefore:

\[
\text{sc}(o, t | C') - (C.o - C.V.o) \leq \text{sc}(o, t | C) \leq \text{sc}(o, t | C') + (C.o - C.V.o).
\]

Summing over all \( t \in Q^V \), replacing the l.h.s. with \( \text{wsc}' \) and the r.h.s. with \( \text{bsc}' \), we obtain:

- \( \text{wsc} \) or \( \text{bsc} \):
  \[
  \text{wsc} = |Q|^V \times |C.o - C.V.o| = f_w(\text{wsc})
  \]
  \[
  \text{bsc} = |Q|^V \times |C.o - C.V.o| = f_b(\text{bsc}).
  \]

We present in Section 8 the generic algorithm that, for a given input query \( Q \) and set of views, filters out the views that are certainly useless for \( Q \), performs the necessary context transposition steps, then may select views, and finally runs one of the threshold algorithms.

### 7.2 Social-aware search

We consider now the social-aware setting. For illustration, we use the collaborative bookmarking applications, which represent a good abstraction for search with a social context (popular examples are applications such as Del.icio.us and Flickr).

For this setting, we revisit the ranking model developed in [20]. Besides objects and attributes, we have a set of users \( U = \{u_1, \ldots, u_n\} \) who can bookmark objects with attributes. Also, users form a social network, seen as an undirected weighted graph: a link between two users \( u_1 \) and \( u_2 \) has a weight, \( \sigma(u_1, u_2) \in [0, 1] \), which could stand for proximity, similarity, affinity, etc. For pairs of users for which an explicit edge (and proximity) is not given, an extended proximity \( \sigma^+(u_1, u_2) \in [0, 1] \) can be computed in the graph by aggregating (e.g., by multiplication) the weights over all the paths connecting \( u_1 \) and \( u_2 \), and taking the maximal aggregated score over them:\footnote{This is reminiscent of how trust or similarity can propagate, if interpreted as transitive measures.}

\[
\sigma^+(u_1, u_2) = \max_{p=(u_1, u_2)} \prod_{o=0}^{k-1} \sigma(u_1, u_{o+1}).
\]

A query context \( C \) consists now of a seeker \( C.I \) (the issuer of the query) and the parameter \( C.o \).

In manner similar to location-aware search, the per-attribute score is a linear combination between the “social location” of the seeker with respect to the taggers of an object and the classic textual score (e.g., tf-idf or BM25).

Unlike the location-aware setting, we do not deal with objects that are “located” at certain points in the search space, but with user-object-attribute tuples \( (\mu, o, t) \), i.e., users \( \mu \) who tagged \( o \) with \( t \). The social component of the score is computed as the sum of the proximity values of taggers of \( o \) with respect to the seeker, while the textual component is the number of taggers who tagged object \( o \) with attribute \( t \):

\[
\text{sc}(o, t | C) = (1 - C.o) \times \sum_{\mu \text{ tagged } o \text{ with } t} \sigma^+(C.I, \mu) + C.o \times TF(o, t),
\]

Transposing the location. For a query \( Q \) and seekers \( C^V.I \) and \( C.I \), let \( u \) be a tagger for which we need to use \( \sigma^+(C.I, u) \) in score bounds. As illustrated in Figure 3 the path with the highest score connecting \( C.I \) to \( u \) may either

1. go through \( C.V.I \), and in that case we have:
   \[
   \sigma^+(C.I, u) = \sigma^+(C.I, C.V.I) \times \sigma^+(C.V.I, u),
   \]
2. not go through \( C.V.I \), and in that case we have:
   \[
   \sigma^+(C.I, u) \leq \sigma^+(C.I, C.V.I)^{-1} \times \sigma^+(C.V.I, u).
   \]

Now, the influence of the social component in the score of \( o \) in \( \text{ans}(V) \) varies inversely with \( C.V.o \). Therefore, the transposition that accounts for the location change should be weighted by its importance in the \( \text{sc}(o, t | C^V) \) formula, which is determined by \( C.V.o \) when \( C.V.o = 1 \), the lower and upper bounds should not be affected by the location change, while when \( C.V.o = 0 \), they should be affected with weight \( \sigma^+(C.I, C.V.I) \) and \( \sigma^+(C.I, C.V.I)^{-1} \) respectively. We can model this by a coefficient function \( c(w, o) \), which applies to a weight \( w \) and value \( o \), and is defined as

\[
c(w, o) = \alpha(1 - w) + w.
\]

(Note that it verifies \( c(w, 0) = w \) and \( c(w, 1) = 1 \), as needed.)
For each object \( o \) of a tuple \((o, wsc, bsc) \in \text{ans}(V)\), we can now derive the following valid bounds for a context \( C' \) defined by the query’s seeker \( C.l \) and the view’s parameter \( C.V.\alpha \).

\[
\begin{align*}
\text{se}(o, t | C') &\geq c(o^+ + C.l, C.V.\alpha) \times wsc = wsc' \\
\text{se}(o, t | C') &\leq c(o^- + C.l, C.V.\alpha) \times bsc = bsc' \quad (7.5)
\end{align*}
\]

Transposing the parameter \( \alpha \). We detail now the transposition for the \( \alpha \) component, from \( C.V.\alpha \) to \( C.\alpha \), yielding valid bounds for the context \( C' \). Here, the transposition depends on the relationship between \( C.\alpha \) and \( C.V.\alpha \), and we obtain the following \( f_o \) and \( f_t \):

1. If \( C.\alpha < C.V.\alpha \), \( f_w(wsc) = \frac{C.\alpha}{C.V.\alpha} \times wsc' \), \( f_b(bsc) = bsc' \)
2. If \( C.\alpha > C.V.\alpha \), \( f_w(wsc) = wsc' \), \( f_b(bsc) = \frac{C.\alpha}{C.V.\alpha} \times bsc' \)

We arrive at these bounds by the following steps. In Eq. (7.4), let

\[
x = \sum_{u \text{ tagged } a \text{ with } t} \sigma^+(C.l, u), \quad y = TF(a, t).
\]

From the intermediary context \( C' \) introduced by the transposition of location, for any tag \( t \in Q.V \) and object \( o \in \text{ans}(V) \), we have:

\[
\begin{align*}
\text{se}(o, t | C') &= (1 - C.V.\alpha) \times x + C.V.\alpha \times y \\
\text{se}(o, t | C) &= (1 - C.\alpha) \times x + C.\alpha \times y 
\end{align*}
\]

Writing \( y \) in terms of \( \text{se}(o, t | C') \) and \( x \) in the first equation we get

\[
y = \frac{\text{se}(o, t | C') - (1 - C.V.\alpha) \times x}{C.V.\alpha}.
\]

Then, by plugging \( y \) into the second equation, we obtain:

\[
\text{se}(o, t | C) = (1 - \frac{C.\alpha}{C.V.\alpha}) \times x + \frac{C.\alpha}{C.V.\alpha} \times \text{se}(o, t | C').
\]

Since we know that \( x \leq \text{se}(o, t | C') \) (due to the fact that \( x \leq y \), by definition), we can derive the following bounds:

\[
\begin{align*}
\text{se}(o, t | C) &\leq \frac{C.\alpha}{C.V.\alpha} \cdot \text{se}(o, t | C') \quad \text{if} \ C.\alpha > C.V.\alpha \\
\text{se}(o, t | C) &\geq \frac{C.\alpha}{C.V.\alpha} \cdot \text{se}(o, t | C') \quad \text{if} \ C.\alpha < C.V.\alpha
\end{align*}
\]

By subtraction, we also have that:

\[
\text{se}(o, t | C) = \text{se}(o, t | C') \times (C.\alpha - C.V.\alpha)(y - x)
\]

Unlike the location-aware setting, we do not have a bound on the difference between \( y \) and \( x \), but we know that \( (y - x) \geq 0 \), and therefore we can obtain:

\[
\begin{align*}
\text{se}(o, t | C) &\leq \text{se}(o, t | C') \quad \text{if} \ C.\alpha > C.V.\alpha \\
\text{se}(o, t | C) &\geq \text{se}(o, t | C') \quad \text{if} \ C.\alpha < C.V.\alpha
\end{align*}
\]

Putting everything together, over all \( t \in Q.V \), we obtain the \( f_w \) and \( f_b \) transposition functions given previously.

### 8. PROCESSING QUERIES USING VIEWS

We put together all the techniques discussed so far into one generic algorithm for answering top- \( k \) queries using views with uncertain scores (Algorithm 5). It starts by transposing the various contexts \( C.V.\) to the one of the input query, and selects a subset of views. Either SR-TA or SR-NRA can then be used to find an answer \((G, P)\), in terms of guaranteed and possible objects for the top- \( k \). We can then use the sampling procedure presented in Algorithm 5 to find a most likely top- \( k \) answer for the input query.

#### Algorithm 5: PROCESSQUERYVIEWS\((V, Q, C, k, r)\)

**Require:** query \( Q \), views \( V \), context \( C \), top \( k \) required, \( r \) rounds

1. for \( V \in V \) do
2. transpose the context \( C.V. \) to \( C.\alpha \)
3. end for
4. \( V \leftarrow \text{view selection on } V \) for \( Q \)
5. \{\( G, P \}\} \leftarrow \text{SR-TA}(Q, k, V) \text{ or SR-NRA}(Q, k, V)
6. \( E = \text{ESTIMATE}(P, k - |G|, r) \)
7. return \( G \cup E \)

In the experiments, we give a detailed comparison between this approach and (i) an existing early-termination algorithm for spatial-aware search, the IR-TREE of [8], and (ii) the early-termination algorithm CONTEXTMERGE of [20], for social-aware search.

**Remark 1.** A significant advantage of the bounds derived in Eq. (7.4) and (7.5) is that they are not dependent on the object: in scores, we only add / subtract or multiply / divide with object-independent values. This gives us the opportunity to use certain implementation “tricks”: instead of recomputing the score bounds of all object in the \( \text{ans}(V) \) lists, we can use these score changes as

1. additive (for location-aware search) or multiplicative (for social-search) coefficients directly in the per-view statistics for view selection,

2. coefficients for the linear programs in Eq. (2.1) and (2.2), during the run of SR-TA or SR-NRA.

**Remark 2.** While we focused in the previous two sections on the basic ranking models for context-aware search used in previous literature, the same reasoning can be adapted to more involved ranking models, as long as they are monotonic relative to both the textual and the context-aware components of scores.

### 9. EXPERIMENTS

We performed our experiments on a single core of a machine i7-860 2.8GHz 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 over spatial data, as we used our own implementation of the competitor algorithm IR-TREE, for which C++ seemed more adapted.

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.
Figure 4: Performance comparison between SR-TA variants over synthetic data with uniform and exponential distribution.

Figure 5: Performance comparison between SR-NRA variants over synthetic data with uniform and exponential distribution.
We compare in Figure 3 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 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-processing on noisier data needs to go deeper in the views to reach termination.

The same measurements were performed for the SR-NRA variants (see Figure 4), in which random- accesses are not allowed. We can observe similar behavior to the SR-TA variants, in terms of running-time. However, we have a much lower relative running-time for SR-NRA^max, less than 0.1 of that of SR-NRA^max. This is due to the fact that the overhead induced by solving the LPs for score bounds is much more noticeable in this case (in the case of SR-TA, it was dominated by the cost of the random accesses).

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^max can have a running-time that is a low fraction of that of SR-TA^max. This is mainly due to the fact that, although inexact, we have aggregated scores pertaining to k 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 [10], our experiments show that even relatively noisy aggregated data can lead to improvements, with reasonable precision.

Finally, we give in Table 4 the overhead introduced by the final refinement step discussed in Section 5 which uses random accesses to refine a result (G, P) to the most informative answer (G’, P). The 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. We can see that, while the possible combinations that are “avoided” increase exponentially with the standard deviation, the overhead introduced by the additional I/O accesses is rather small, in the range 3% – 13%.

| 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.

| Sel. | Std | Overhead | |G| | |G| | |P| | |P| | Δ |
|------|-----|----------|---|---|---|---|---|
| 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 × 10^59  |
| 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-NRA^max with final refinement versus SR-NRA^max without refinement, for k=100 and exponential distribution.
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 [20]. In short, depending on the value of $\alpha$, 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 TA, 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 6 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 exclusively on the social component of the score. This is due to the fact that the bounds computed by Eq. (7,3) 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 $\alpha$ 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).

10. MAIN RELATED WORK

Efficient processing of top-k queries is an important research topic in recent IR literature, leading to advancements in both data structures and query processing algorithms. The most common data structure is the inverted index file (for a general survey on indexing for top-k processing, see [27]), over which a key challenge is to optimize response time. Regarding algorithms, among the most widely cited and used are the early-termination threshold algorithms TA and NRA of [11], which can 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 [13] and the references therein.

The use of precomputed results, either as previous answers to queries [9] or as cached intersection lists [16], has been identified as an important direction for efficiency. A linear programming formulation over score information is first introduced in [9] and extended in [16].

In [21], 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. Another general formulation of ranking in probabilistic databases is presented in [17].

In the area of location-aware retrieval, Cong et al. [8] introduce the concept of L&T 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 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 [5], or the k-best objects scored by the features in their neighborhood [19], or the top-k objects in a given query rectangle [7]. Various approaches for combining textual inverted lists and spatial indexes for keyword retrieval were also studied in [7].

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 [20] 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 [6], personalization based on a similarity network is shown to outperform other personalization approaches and the non-personalized search. An architecture for social data management and a framework for information discovery and presentation are given in [23]. Other studies on personalizing search results in collaborative bookmarking applications can be found in [22].

11. CONCLUSION AND FUTURE WORK

We formalize and study in this paper the problem of context-aware top-k processing based on uncertain precomputed results, in the form of views over the data. This problem is motivated on one hand by search applications in which query results depend on a context, and any result caching or pre-computation mechanism needs to perform certain transformations – what we call a context transformation – in order to be able to answer new queries, which may pertain to new contexts. On the other hand, even in context-agnostic
search scenarios, some of the most common threshold algorithms, such as NRA, for more efficiency, may output results with score ranges (i.e., uncertain scores) instead of exact scores.

We introduce the query semantics needed for dealing with objects of uncertain scores and describe two algorithms, SR-TA and SR-NRA that support this semantics and are sound and complete, i.e., they output what we call the most informative result: (i) all the guaranteed objects, and (ii) all and only the objects that may appear in the top-k in some data instance. We also consider optimizations for SR-TA and SR-NRA, based on selecting some(few) most promising views, instead of using the complete, potentially very large, set of views. 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 data, for spatial and social search, 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 area of applications.

Importantly, our algorithms provide a one-size-fits-all solution for many search applications that are context-dependent, with the only application-dependent aspect being the context transposition.

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 in which it relies only on a subset of the remaining views, possibly in an incremental manner. The study of more precise definitions of classes of views that guarantee instance optimality is another avenue of further research.

With respect to applications, other context-aware scenarios and application-dependent context transpositions, as well as other ranking models, can 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.

12. REFERENCES


APPENDIX

A. PROOF OF LEMMA 1

PROOF. We remind the fact that any algorithm only needs to use the views contained in the connected component to which Q belongs, from the attribute-query graph introduced in Section 5. We will denote the set of views contained in this connected component, except the query Q, as \( V_C(Q) \). We also denote as *relevant combinations of views* the set \( V(Q) = \{V_1(Q), \ldots, V_n(Q)\} \), composed of distinct (but not necessarily disjoint) sets \( V_i(Q) \subseteq V_C(Q) \) of views which can be used to compute \( [wsc, bsc] \) intervals for any object \( o \).

We start by proving that the no view selection variant of the two algorithms can perform unnecessary accesses.

Let us consider the following instance of the data and the set of views, \( V \). Assume that for a given query \( Q \), the set of relevant combinations \( V(Q) \) contains all the views in \( V_C(Q) \). Furthermore, the set of relevant combinations consists of two disjoint sets \( V_1(Q) \) and \( V_2(Q) \). We also assume w.l.o.g. that \( |V_1(Q)| = |V_2(Q)| = s \). The case where they have different sizes follows a similar reasoning, but would clutter the presentation unnecessarily.

Let us construct the following database: all the objects in the views of \( V_1(Q) \) are present in \( V_2(Q) \) at the same depths. Each such object \( o \) is constructed to ensure that \( wsc_2(V_1(Q)) \geq wsc_2(V_2(Q)) \) and \( bsc_2(V_1(Q)) \leq bsc_2(V_2(Q)) \), i.e., that the best bounds are always obtained from \( V_1(Q) \).

Now, let’s assume that SR-TA\(_{nosel}^\text{1} \) or SR-NRA\(_{nosel}^\text{1} \) stop at depth \( d \), encountering a \( \leq sd \) distinct objects, and returning \( P \) and \( G \) as desired. They have thus performed the following number of accesses: (i) \( 2sd \) sequential accesses for SR-NRA\(_{nosel}^\text{1} \) and SR-TA\(_{nosel}^\text{1} \), and (ii) \( (2sd - 1)\alpha \) random accesses for SR-TA\(_{nosel}^\text{1} \). The accesses to the \( V_1(Q) \)’s bestscore lists would be \( h_1 \leq sa \), and \( h_2 \leq sa \) to \( V_2(Q) \)’s for a total of \( h_1 + h_2 \).

Then, algorithms SR-TA\(_{nosel}^\text{1} \) or SR-NRA\(_{nosel}^\text{1} \) that only accesses the views in \( V_1(Q) \) would encounter the same number \( \alpha \) of distinct items, since \( V_1(Q) \) and \( V_2(Q) \) have the same objects at the same depths. It would also stop at the same depth \( d \) as the no selection algorithms, as it would obtain the same bounds for the \( a \) objects, and the same value for the threshold. Hence SR-TA\(_{nosel}^\text{1} \) and SR-NRA\(_{nosel}^\text{1} \) would need only \( sd \) sequential accesses, while SR-TA\(_{nosel}^\text{1} \) would only need \( (sd - 1)\alpha \) random accesses. The same reasoning holds for accesses into bestscore lists: both algorithms need only the \( h_1 \) accesses detailed above.

We turn next to proving that the selection variants of the two algorithms can perform unnecessary accesses.

Let us construct the following database, for a given \( k \) and query \( Q \). The set \( V(Q) \) of relevant views is composed, as above, of two disjoint sets of views \( V_1(Q) \) and \( V_2(Q) \) that both can give valid bounds for \( Q \). Again, assume w.l.o.g. \( |V_1(Q)| = |V_2(Q)| = s \).

Then, on the first \( k \) positions in \( V_1(Q) \) we insert \( k \) objects s.t., for any such object \( o \), \( wsc_2(V_1(Q)) = \delta \) and \( bsc_2(V_1(Q)) = \delta + 2 \). In the following \( k \) positions, we insert objects \( x \) s.t. \( wsc_2(V_1(Q)) = \delta - 1 \) and \( bsc_2(V_1(Q)) = \delta + 1 \). The rest of the objects, denoted \( * \), are constructed s.t. \( wsc_2(V_1(Q)) = \delta - 1 \).

For \( V_2(Q) \), the same \( k \) objects are inserted s.t. \( wsc_2(V_2(Q)) = \delta - 1 \) and \( bsc_2(V_2(Q)) = \delta + 1 \). In the next \( k \) positions, we insert the same objects \( x \) as above, s.t. \( wsc_2(V_2(Q)) = \delta - 1 \) and \( bsc_2(V_2(Q)) = \delta \). The rest of the objects \( * \) are constructed as above.

We analyze now the accesses needed for the algorithms using all views. Both algorithms will stop at depth \( k \), since the threshold value is \( \tau = \delta \) and the lowest worstscore of the \( k \) items is \( \delta \). Hence, the first \( k \) objects are in \( G \) and \( P = \emptyset \). SR-NRA\(_{nosel}^\text{1} \) would perform \( 2sk \) sequential accesses in the worstscore lists and \( 2sk \) accesses to the bestscore lists. SR-TA\(_{nosel}^\text{1} \) would perform the same number of sequential accesses as TA and also \( (2s - k)k \) random accesses.

SR-NRA\(_{nosel}^\text{1} \), using either \( V_1(Q) \) or \( V_2(Q) \), would only be able to stop at depth \( 2k \), thus performing \( 2sk \) sequential accesses in the worstscore lists and \( 2sk \) accesses in bestscore lists. It would also return \( P \) containing all \( 2k \) objects, as the bestscores of the bottom \( k \) objects are greater than the worstscores of the top \( k \) objects, and \( G = \emptyset \). Hence, a score refinement step would need to be performed. If we still assume that only sequential accesses are allowed, the refinement step would need another \( k \) sequential accesses into the “opposite” set of views, to establish that \( P = \emptyset \) and \( G \) is composed of the objects in the first \( k \) positions. Hence it would need \( ks \) more sequential accesses than SR-NRA\(_{nosel}^\text{1} \).

For SR-TA\(_{nosel}^\text{1} \), the stopping condition would also hold at depth \( 2k \). The same number of \( 4sk \) sequential accesses would be needed, and \( 2k(s - 1) \) random accesses. For refining the scores and returning the sets \( P \) and \( G \), another \( 2ks \) accesses to the other views would also be needed. Hence it would need \( k(2s - 1) \) more random accesses than SR-TA\(_{nosel}^\text{1} \). □