Exact and Approximate Generic Multi-criteria Top-k Query Processing

Mehdi Badr, Dan Vodislav

Exact and approximate generic multi-criteria top-k query processing

Mehdi Badr[1,2] and Dan Vodislav[1]

[1] ETIS, ENSEA - University of Cergy-Pontoise - CNRS, Cergy, France
[2] TRIMANE, 57 rue de Mareil, Saint-Germain-en-Laye, France

Abstract. Many algorithms for multi-criteria top-k query processing with ranking predicates have been proposed, but little effort has been directed toward genericity, i.e. supporting any type of access to the lists of predicate scores (sorted and/or random), or any access cost settings. In this paper we propose a general approach to exact and approximate generic top-k processing. To this end, we propose a general framework (GF) for generic top-k processing, able to express any top-k algorithm and present within this framework a first comparison between generic algorithms. In previous work, we proposed BreadthRefine (BR), a generic algorithm that considers the current top-k candidates as a whole instead of focusing on the best candidate for score refinement, then we compared it with specific top-k algorithms. In this paper, we propose two variants of existing generic strategies and experimentally compare them with the BR breadth-first strategy, showing that BR leads to better execution costs. We also extend the notion of θ-approximation to the GF framework and present a first experimental study of the approximation potential of top-k algorithms on early stopping.

Keywords: Top-k query processing, ranking, multi-criteria information retrieval.

1 Introduction

We address the problem of top-k multi-criteria query processing, where queries are composed of a set of ranking predicates, each one expressing a measure of similarity between data objects on some specific criterion. Unlike traditional Boolean predicates, similarity predicates return a relevance score in a given interval. The query also specifies an aggregation function that combines the scores produced by the similarity predicate of each criterion. Query results are ranked following the global score and only the best k ones are returned.

Ranking predicates acquired an increasing importance in today's data retrieval applications, especially with the introduction of new, weakly structured data types: text, images, maps, etc. Searching such data requires content-based information retrieval (CBIR) techniques, based on predicates measuring the similarity between data objects, by using content descriptors such as keyword sets, image descriptors, geographical coordinates, etc. We consider here the case of *expensive* ranking predicates over data objects, whose specificity is that their evaluation cost dominates the cost of the other query processing operations.

select * **from** Object o
order by $\mathcal{F}(p_1(o), ..., p_m(o))$
limit k

select * **from** Monument m
order by *near*(m.address, here()) +
similar(m.photo, myPhoto) +
ftcontains(m.descr, 'Renaissance sculpture')
limit 1

Fig. 1. General form and example of a top-k query

The general form of the top-k queries that we consider is expressed in Figure 1. The query asks for the k best objects following the scores produced by m ranking predicates p_1, ..., p_m, aggregated by a monotone function $\mathcal{F}$.

Figure 1 also presents a query example coming e.g. from a smartphone touristic application, where the visitor of a Renaissance monument, after finishing the visit, searches for another similar monument (the "best" one) on three criteria: *near* to his current location, containing a *similar detail* to some picture taken with the smartphone, and exposing *Renaissance artworks, preferably sculptures*. Here, the aggregate function is a simple sum.

As in this example, expensive ranking predicates come often from the evaluation of similarity between images, text, locations and other multimedia types, whose content is described by numerical vectors. This results in expensive searches in highly dimensional spaces, based often on specific multidimensional index structures [3]. Note that most of the ranked predicates in this case come from binary predicates $sim(o_1, o_2)$ evaluating similarity between objects, transformed into unary ranked predicates $p(o) = sim(o, q)$ evaluating the similarity with a query object q.

In many cases, predicates are evaluated by distant, specialized sites, that provide specific web services, e.g. map services evaluating spatial proximity, photo sharing sites allowing search of similar images, specialized web sites proposing rankings for hotels, restaurants, etc. Internet access to such services results into expensive predicate evaluation by distant, independent sites. Moreover, the control over predicate evaluation is minimal most of the time, reduced to the call of the provided web service.

For each query, a ranking predicate may produce a score for each object. Following, we call *a source* the collection of scores produced by a ranking predicate for the set of data objects. The list of scores may be produced e.g. by accessing a local index structure that returns results by order of relevance. We consider here the general case, where the access to the scores of a source is limited to sorted and/or random access. This allows three possible types for a source S:

- *S-source*: sorted access only, through the operator *getNext(S)* returning the pair (o, s) containing the identifier o of the object with the next highest score s.
- *R-source*: random access only, through the operator *getScore(S, o)* returning the score of a given object o.
- *SR-source*: a source with both sorted and random access.

The general idea of a top-k algorithm is to avoid computing all the global scores, by maintaining a list of candidate objects and the interval $[L, U]$ of possible global

S_1 (S)	S_2 (SR)	S_3 (R)
(o_2, 0.4)	(o_3, 0.9)	(o_1, 0.9)
(o_1, 0.3)	(o_1, 0.2)	(o_2, 0.7)
(o_4, 0.25)	(o_4, 0.15)	(o_3, 0.8)
(o_3, 0.2)	(o_2, 0.1)	(o_4, 0.6)

Access	Retrieved	*candidates*	U_{unseen}
		$\emptyset$	3.0
S_1/S	(o_2, 0.4)	{(o_2, $[0.4, 2.4]$)}	2.4
S_2/S	(o_3, 0.9)	{(o_2, $[0.4, 2.3]$), (o_3, $[0.9, 2.3]$)}	2.3
S_2/R	(o_2, 0.1)	{(o_2, $[0.5, 1.5]$), (o_3, $[0.9, 2.3]$)}	2.3
S_3/R	(o_3, 0.8)	{(o_2, $[0.5, 1.5]$), (o_3, $[1.7, 2.1]$)}	2.3
S_2/S	(o_1, 0.2)	{(o_2, $[0.5, 1.5]$), (o_3, $[1.7, 2.1]$), (o_1, $[0.2, 1.6]$)}	1.6

Fig. 2. Examples of sources and query execution for the query example

scores for each of them. The initial interval of a candidate is obtained by aggregating the minimum/ maximum source scores.

The monotonicity of the aggregation function ensures that further source accesses always decrease the upper bound U and increase the lower bound L. The algorithm stops when the score of the best k candidates cannot be exceeded by the other objects anymore.

Figure 2 presents a possible execution for the example query in Figure 1. We suppose S_1 is an S-source, S_2 an SR-source, S_3 an R-source; scores are presented in descending order for S/SR sources and by object identifier for R-sources. Local scores belong to the $[0, 1]$ interval in this example, so the initial global score interval is $[0, 3]$ for all objects.

We note *candidates* the set of candidates and U_{unseen} the maximum score of unseen objects (not yet discovered in some source). Initially, *candidates* = $\emptyset$ and U_{unseen} = 3.

- A sorted access to S_1 retrieves (o_2, 0.4), so o_2's global score interval becomes $[0.4, 2.4]$. Also U_{unseen} becomes 2.4 because further scores in S_1 cannot exceed 0.4.
- Then, a sorted access to S_2 retrieves (o_3, 0.9). This adds a new candidate (o_3), lowers U_{unseen} to 2.3 (further S_2 scores cannot exceed 0.9), but also lowers the upper bound of o_2 to 2.3, because the maximum score of S_2 is now 0.9.
- Next, a random access to S_2 for o_2 retrieves (o_2, 0.1). This changes only the global score interval of o_2.
- A random access to S_3 for o_3 retrieves (o_3, 0.8) and changes the global score interval of o3.
- A sorted access to S_2 retrieves (o_1, 0.2). This adds a new candidate (o_1), lowers U_{unseen} to 1.6, but does not lower the maximal global score of the other candidates because o_2 and o_3 already know their S_2 scores.

The minimum global score of o_3 exceeds now both U_{unseen} and the maximum global score of all the other candidates and the execution stops since o_3 is surely the best (top-1) object.

2 Related work and contribution

A large spectrum of top-k query processing techniques [11] has been proposed at different levels: query model, access types, implementation structures, etc. We consider here the most general case, of simple top-k selection queries, with expensive access to sources, limited to individual sorted/random probes, without additional information about local source scores/objects, and out of the database engine.

This excludes from our context join queries [17, 10] or interaction with other database operators for query optimization [13, 10, 12]. We consider sequential access only, parallel processing is out of the scope of this paper. We exclude also approaches such as TPUT [5], KLEE [16] or BPA [1], able to get several items at once, or having statistical information available about scores, or having also the local rank. Algorithms such as LARA [14], that optimize the management of the candidate list, are orthogonal to our approach for expensive predicates, which focuses on source access.

In this context, top-k algorithms proposed so far fit with the general method presented in the example of Figure 2 and propose their own heuristic for deciding the next access to a score source. However, most algorithms focus on specific source types and cost settings.

Algorithms such as *NRA*[7] (No Random Access) and *StreamCombine*[9] consider only S-sources. NRA successively consults all the sources in a fixed order, while StreamCombine selects at each step the next access based on a notion of *source benefit*.

Other algorithms consider only SR-sources. The best known is *TA*[7] (Threshold Algorithm), which consults sorted sources in a fixed order (like NRA), but fully evaluates the global score of each candidate through random access to the other sources. The algorithm stops when at least k global scores exceed U_{unseen}. Among the extensions of TA we cite *QuickCombine*[8], which uses the same idea as StreamCombine to select the next sorted source to probe, and *TAz*[4], which considers an additional set of R-sources besides the SR-sources. *CA*[7] (Combined Algorithm) is a combination of TA with NRA that considers random accesses being h times more expensive than sorted ones. It reduces the number of random probes by performing h sorted accesses in each source before a complete evaluation of the best candidate by random probes.

Also supposing cost asymmetry, a third category of algorithms considers one cheap S-source (providing candidates) and several expensive R-sources. *Upper*[4, 15] focuses on the candidate with the highest upper bound U and performs a random probe for it, unless $U < U_{unseen}$, in which case a sorted access is done. The choice of the next R-source to probe is based on a notion of source benefit, dynamically computed. *MPro*[6] is similar to Upper, but fixes for all the candidates the same order for probing the R-sources, determined by sampling optimization.

Surprisingly, little effort has been made towards generic top-k processing, i.e. adapted to any combination of source types and any cost settings. To our knowledge, besides our BreadthRefine proposal described below, *NC*[19] (Necessary Choices) is the only other generic approach, however limited to the case of results *with complete scoring*. NC proposes a framework for generic top-k algorithms, a strategy SR that favors sorted accesses, and a specific algorithm SR/G that uses sampling optimization to find the parameters that produce the best fit given the source settings.

Approximate top-k processing has been considered in several approaches, the most usual one being the approximation by early stopping, i.e. considering the current top-k objects at some point during the execution as an approximate result. Since early stopping comes with no guarantees on the quality of the result, several constraints providing such guarantees have been considered. For instance, a variant of the TA algorithm, called TA$_\theta$ [7], defines an approximation parameter $\theta > 1$ and the θ-approximation of the top-k result as being a set K_a of k objects such that $\forall x \in K_a, \forall y \notin K_a$, $\theta \times score(x) \geq score(y)$ (global and local scores are considered to belong to the [0,1] interval). The intuition behind this condition is that the ratio between the score of the best missed object in the approximate result (best false negative) and that of the worst false positive cannot exceed θ. To obtain a θ-approximation, TA$_\theta$ simply changes the threshold condition: the algorithm stops when at least k objects have a global score $\geq U_{unseen}/\theta$, i.e. TA$_\theta$ is equivalent to an early stopping of the TA algorithm.

Other approximation algorithms for top-k selection queries are proposed in [18], for S-source algorithms, or in the *KLEE* system [16] for top-k processing in distributed environments. Note that [18] is based on dropping candidates that have low probability to be in the top-k and provides probabilistic guarantees for the result, but requires knowledge about score distribution in sources.

In previous work, we have proposed *BR* (BreadthRefine) [2], a generic algorithm that uses a breadth-first strategy for top-k processing in a larger context than NC, i.e. with incomplete scoring. The BR strategy considers the current top-k as a whole to be refined, while all the other proposed strategies focus on the best candidate. BR has been compared to algorithms of the three categories mentioned above and proved that it successfully adapts to their specific settings, with better cost.

In this paper, we address exact and approximate multi-criteria top-k query processing at a general level, proposing generalizations of existing algorithms to the generic case and aiming at a comparison of algorithm strategies. More precisely, our contributions are the following:

- A general framework *GF* for generic top-k multi-criteria query processing, that allows expressing any top-k algorithm of our context, thus providing a basis for comparative analysis and generalization.
- The BR algorithm is generic (adapted to any combination of source types and any cost settings), but it was only compared to specific top-k algorithms, since the only other generic approach, introduced by NC, is hardly comparable with BR. As further detailed in Section 3, the difficulty to compare with NC comes mainly from the fact that, unlike BR and most top-k algorithms, NC is not fully heuristic and strongly depends on a sampling optimization phase. We propose here new, comparable generic variants of the BR, NC and CA algorithms and experimentally compare these generic strategies, showing that BR leads to better execution costs.
- A generalization of θ-approximation computing in the context of GF, and a first experimental study of the ability of top-k multi-criteria algorithms to produce good approximate results on early stopping, showing that the BR strategy comes with a better approximation potential.

We do not directly address here algorithm optimality issues. Fagin *et al.* demonstrate in [7] that NRA and TA algorithms are instance optimal, i.e. for any database

instance, no top-k algorithm can improve the execution cost with more than a constant factor. They also show that algorithms based on a dynamic choice of the next source to access (such as BR or, more generally, algorithms expressed in GF) may not be instance optimal, although they may have in practice better execution costs. Even if BR and the other generic algorithms we consider are not instance optimal, our goal is to experimentally demonstrate that the BR strategy leads to better performances. Note however that, as shown in [7], BR could be adapted to become instance optimal by adding source accesses that guarantee every source to be accessed at least once every C steps, for some constant C.

The rest of the paper is organized as follows: the next section introduces the generic framework for top-k multi-criteria processing, then proposes and compares in this context new generic variants for BR, NC and CA. Section 4 presents our approach for top-k approximation in the general framework, then we report experimental results and end with conclusions.

3 Generic top-k framework and algorithms

We propose *GF*, a *generic framework* for multi-criteria top-k processing (Figure 3). GF provides a common, general form for expressing any top-k algorithm in our context. It facilitates comparison between top-k algorithms and strategies expressed in this common form. For instance, we benefit here from this common framework in the description of new variants of existing algorithms (NC and CA), compared then with our BR approach. Another major benefit of GF is that new properties expressed and proved in this general framework are true for any top-k algorithm - for instance, the θ-approximation properties presented in Section 4.

As in the example of Figure 2, GF considers a top-k algorithm as a sequence of accesses to the sources, that progressively discover scoring information about data objects. The input parameters are the query q and the set of sources S. Query q specifies the number k of results to return and the monotone aggregation function $\mathcal{F}$, while the set of sources S materializes the scores returned by the query's ranking predicates.

In GF, algorithms maintain *a set of candidates* (initially empty) with their interval of scores, *the threshold* U_{unseen} (initialized with the aggregation of the maximum scores max_j of the sources), and possibly other local data structures.

Notations

- For a candidate c, we note $[L(c), U(c)]$ its current interval of scores.
- We note $\mathcal{U}_k$ (respectively $\mathcal{L}_k$) the current subset of k candidates with the best k upper (lower) bound scores[3].
- We note U_k the current k-th highest upper bound score among the candidates, i.e. $U_k = min_{c \in \mathcal{U}_k}(U(c))$, respectively L_k the current k-th highest lower bound score, $L_k = min_{c \in \mathcal{L}_k}(L(c))$.
- We note $\chi \in \mathcal{U}_1$ the candidate with the current best upper bound score.

[3] With random selection among candidates with the same score if necessary

GF (q, $\mathcal{S}$)
  candidates $\leftarrow \emptyset$; $U_{unseen} \leftarrow \mathcal{F}(max_1, ..., max_m)$; ... *//other local variables*
  repeat *//choice between sorted or random access*
    if *SortedAccessCondition()* **then** *//sorted access*
      $S_j \leftarrow$ **BestSortedSource()** *//choice of a sorted source*
      (o, s) $\leftarrow$ getNext(S_j) *//sorted access to the selected source*
      Update *candidates*, U_{unseen} and other local variables
    else *//random access*
      $c \leftarrow$ **ChooseCandidate()** *//choice of a candidate*
      $S_j \leftarrow$ **BestRandomSource**(c) *//choice of a random source*
      $s \leftarrow$ getScore(S_j, c) *//random access to the selected source*
      Update *candidates* and other local variables
    endif
  until *StopCondition()*
  return *candidates*

Fig. 3. The GF generic top-k framework

Note that the monotonicity of the aggregation function guarantees that the threshold and the upper bound of candidate scores only decrease, while their lower bound only increase during the algorithm.

One source access is performed at each iteration, the access type being decided by the *SortedAccessCondition* predicate. In the case of a sorted access, a source S_j is chosen by the **BestSortedSource** function, then is accessed through *getNext*. The returned object-score couple is used to update the threshold, the set of candidates and the local variables. The retrieved object is added/updated in the *candidates* set and objects not yet retrieved in S_j update their upper bounds.

Update also includes *the discarding of non-viable candidates*. A candidate c with $U(c) < L_k$ is called non-viable because it will never be in the top-k result since at least k candidates surely have better scores.

In the case of a random access, the **ChooseCandidate** function selects a candidate c, then **BestRandomSource** gives a random source to probe for it. After the random access through *getScore*, the *candidates* set and local variables are updated (among candidates, only c changes).

The end of the algorithm is controlled by the generic *StopCondition* predicate, which depends on the type of top-k result expected (e.g. with complete or incomplete scoring). The earliest end is obtained with predicate

$$StopCondition \equiv (|candidates| = k \wedge L_k \geq U_{unseen}) \tag{1}$$

i.e. only k candidates are viable and there is no viable unseen object. Since this result may have incomplete scoring, additional conditions are necessary to ensure properties such as ordering or complete scoring of the results.

It is simple to demonstrate that this condition is necessary and sufficient for a correct top-k result. Sufficiency is trivial, the k remaining candidates form a correct top-k, because their scores are at least L_k, while the score of non viable candidates and that of

unseen objects is $\leq L_k$. Necessity comes from the fact that if condition (1) is not true, either $|candidates| < k$ (and then we do not have k candidates to form the result), or $|candidates| > k$ (and then any of the viable candidates still may have a final score that corresponds to a top-k object), or $L_k < U_{unseen}$ (and then an unseen object may belong to the top-k).

It is easy to see that any top-k algorithm in our context can be expressed in GF. Indeed, for a given query and set of sources, each algorithm is equivalent to the sequence of accesses to the sources it produces, which can be obtained with a sequence of decisions about the access type, the source and the candidate for random probes.

Note that this is not true for instance with the NC framework [19], in which one chooses first a candidate among the k highest upper bound scores, then a source in which the candidate has not been yet retrieved. This is not compatible with algorithms that fix the order of accessing sources, such as NRA: a source in which candidates with the current k highest upper bound scores have been already found cannot be selected for the next step.

As an example, the NRA algorithm can be expressed in GF with *SortedAccessCondition* $\equiv$ *true* (only S-sources), a local variable keeping the last accessed source and a function **BestSortedSource** returning the next source in a round robin order.

Note that algorithms with SR-sources only, like TA, may avoid maintaining interval scores, because the global score of each candidate is immediately computed; however, this optimization is not relevant in our context, where cost is given by the access to the sources and not by the updates to local data structures.

Given its ability to express any top-k algorithm, the GF framework is a valuable tool for comparing top-k strategies. Following, we express in GF and compare three generic algorithms: a new variant of BR and new, generic and comparable variants of the NC and CA algorithms.

3.1 BreadthRefine

BreadthRefine (BR) [2] proposes a generic algorithm framework that can be instantiated to several variants. The main idea of the BR strategy is to maintain the set of current top-k candidates $\mathcal{U}_k$ as a whole, instead of focusing on the best candidate χ, which is the common approach.

BR was successfully compared with state of the art non-generic algorithms in their specific settings. We complete here this comparison by considering also two other generic top-k strategies, adapted for that purpose to our context.

The BR framework can be expressed in the more general GF framework by instantiating *SortedAccessCondition* and **ChooseCandidate** to realize the BR strategy.

- *SortedAccessCondition* $\equiv$ ($|candidates| < k$ **or** $U_{unseen} > U_k$ **or** *CostCondition()*)
  The *SortedAccessCondition* in the BR strategy combines three conditions: a sorted access is scheduled if (i) there are not yet k candidates, or (ii) an unseen object could belong to the current top-k $\mathcal{U}_k$ ($U_{unseen} > U_k$), or (iii) a generic *CostCondition* favors sorted access in the typical case where a random access is more expensive than a sorted one.

Condition (ii) targets the decrease of U_{unseen} through sorted accesses and is the heart of the BR strategy for sorted sources. The common strategy for sorted access focuses only on the best candidate χ, and to be sure that χ (and not some unseen object) has the best upper score, a sorted access is scheduled if $U_{unseen} > U(\chi)$ to decrease U_{unseen} below $U(\chi)$. The BR strategy focuses on the whole current top-k: it maintains the whole $\mathcal{U}_k$ free of unseen objects, by scheduling a sorted access if $U_{unseen} > U_k$.

– The BR strategy is completed by the **ChooseCandidate** function for refinement by random probes. All the existing algorithms facing this choice systematically select the best current candidate χ. Instead, the BR strategy maintains the k best candidates as a whole by first selecting the least refined candidate in $\mathcal{U}_k$.

BR considers top-k with incomplete scoring, thus *StopCondition* is given by (1).

BR-Cost* Several instantiations of the BR framework have been proposed in [2]. The one producing the best costs is *BR-Cost*, that fully implements the BR strategy and uses a *CostCondition* inspired from CA: if r is the ratio between the average costs of random and sorted accesses, then successive random probes must be separated by at least r sorted accesses.

In BR-Cost, **BestSortedSource** and **BestRandomSource** adopt a benefit-oriented strategy, inspired by StreamCombine [9] for choosing a sorted source, or by algorithms with controlled random probes such as Upper [4] for random access.

– For a sorted access, the benefit of source S_j is $Bs_j = (\partial\mathcal{F}/\partial S_j) \times N_j \times \delta_j / C_s(S_j)$, where $(\partial\mathcal{F}/\partial S_j)$ is the weight of S_j in the aggregation function, N_j the number of candidates in $\mathcal{U}_k$ not yet seen in S_j, δ_j the expected decrease of the score in S_j and $C_s(S_j)$ the cost of a sorted access in S_j. Since $(\partial\mathcal{F}/\partial S_j)$ cannot be computed for any monotone function $\mathcal{F}$, we consider here, for simplicity, *only the case of weighted sum*, in which $(\partial\mathcal{F}/\partial S_j) = coef_j > 0$, where $coef_j$ is the coefficient corresponding to source S_j in the weighted sum. The value of δ_j can be obtained, e.g. by making one access ahead, with negligible extra cost.
  The intuition behind this formula is that the benefit measures the potential refinement of the candidates score intervals, relative to the access cost. The sorted access to S_j refines not only the score interval of the retrieved object, but also that of objects not yet found in S_j; for these objects the upper bound decreases by $coef_j \times \delta_j$. This formula, borrowed from StreamCombine [9], only considers the N_j candidates of the current top-k not yet found in S_j.
– For a random access, the benefit of source S_j is $Br_j = coef_j \times (crtmax_j - min_j)/C_r(S_j)$, where $crtmax_j$ and min_j are respectively the current maximum score and the minimum score in S_j and $C_r(S_j)$ is the cost of a random probe in S_j. Note that $crtmax_j$ decreases in SR-sources (after sorted accesses), but remains constant (equal to max_j) in R-sources. Note also that $coef_j \times (crtmax_j - min_j)$ measures the reduction of the candidate's score interval size after a random probe in S_j, i.e. here also, the benefit expresses the refinement of the score interval of the accessed object, relative to the access cost.

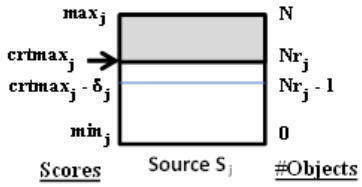


Fig. 4. Scores in a sorted source S_j

We propose here ***BR-Cost*****, an improved variant of BR-Cost, using a different method for estimating r, in this case as a *ratio of benefits* instead of a ratio of access costs. We measure r as the ratio between the average benefit of making a sorted access vs a random one.

As for **BestSortedSource** and **BestRandomSource**, we consider the benefit of an access to a source as being related to its impact on the evolution toward the final top-k, measured by the decrease of the size of the interval of scores of the candidates. More precisely, the benefit of an access to a source is defined as the ratio between the refinement produced on *all* the candidate score intervals and the cost of that access.

Note that this definition corresponds to that used by **BestRandomSource** for random access, because only one candidate is impacted by a random probe, but generalizes the benefit used by **BestSortedSource**, by considering the decrease of score intervals for all the candidates, not only for those of the current top-k. This corresponds to an uniform model for the benefit of accessing any type of source and is more adapted for computing an average benefit. This approach also favors the comparison with the NC strategy.

Consider the case of a S-source S_j in Figure 4 at the moment when the current score is $crtmax_j$ and Nr_j objects have not been yet accessed. A sorted access to S_j refines the score of the retrieved object, but also produces a decrease δ_j of $crtmax_j$ that affects the upper bound of the remaining $Nr_j - 1$ objects. For the retrieved object, the width of the score interval decreases with $coef_j \times (crtmax_j - min_j)$. For each one of the remaining $Nr_j - 1$ objects, the upper bound decreases with $coef_j \times \delta_j$.

In conclusion, the benefit of a sorted access to S_j is:

$$B_s(S_j) = coef_j \times (crtmax_j - min_j + (Nr_j - 1) \times \delta_j)/C_s(S_j) \tag{2}$$

Benefit varies in time; if δ_j does not vary much, benefit globally decreases because $crtmax_j$ and Nr_j decrease. We approximate the average benefit by considering $\delta_j \approx (max_j - min_j)/N$, $crtmax_j \approx (max_j - min_j)/2$ and $Nr_j \approx N/2$:

$$\overline{B_s}(S_j) \approx coef_j \times (max_j - min_j)/C_s(S_j) \tag{3}$$

Note that the instant benefit $B_s(S_j)$ may also be computed at any moment if the total number of objects in the database is known or can be estimated. The instant benefit could be used e.g. as an alternative value for **BestSortedSource**, or for computing a variable ratio r in the BR-Cost* algorithm. Following, we only consider a fixed ratio r, based on the average source benefit.

Benefit for a random access is computed in a similar way, but in this case only the score interval of the selected candidate changes. If S_j is a SR-source, the benefit, respectively the average benefit of a random access are:

$$B_{rs}(S_j) = coef_j \times (crtmax_j - min_j)/C_r(S_j) \tag{4}$$

$$\overline{B_{rs}}(S_j) \approx coef_j \times (max_j - min_j)/2C_r(S_j) \tag{5}$$

For a R-source $crtmax_j = max_j$ all the time, therefore

$$B_r(S_j) = \overline{B_r}(S_j) = coef_j \times (max_j - min_j)/C_r(S_j) \tag{6}$$

The global benefit SB (RB) of processing sorted (random) accesses is defined as the sum of average benefits of the sources allowing this kind of access.

$$SB = \sum_{S_j \in \mathcal{S}_S \cup \mathcal{S}_{SR}} \overline{B_s}(S_j)$$

$$RB = \sum_{S_j \in \mathcal{S}_R} \overline{B_r}(S_j) + \sum_{S_j \in \mathcal{S}_{SR}} \overline{B_{rs}}(S_j)$$

where $\mathcal{S}_S$, $\mathcal{S}_R$ and $\mathcal{S}_{SR}$ are respectively the disjoint sets of S-sources, R-sources and SR-sources.

In conclusion, after developing the terms of SB and RB following formulas (3), (5) and (6) above, the access ratio r used by BR-Cost* becomes:

$$r = SB/RB = \frac{\sum_{S_j \in \mathcal{S}_S \cup \mathcal{S}_{SR}} \frac{A_j}{C_s(S_j)}}{\sum_{S_j \in \mathcal{S}_R} \frac{A_j}{C_r(S_j)} + \sum_{S_j \in \mathcal{S}_{SR}} \frac{A_j}{2C_r(S_j)}} \tag{7}$$

where $A_j = coef_j \times (max_j - min_j)$ is the amplitude of the interval produced by S_j in the aggregated score.

Example As an example, we present in Figure 5 the first steps of BR-Cost* for the query in Figure 1 over the sources in Figure 2. We have $\mathcal{S}_S = \{S_1\}, \mathcal{S}_R = \{S_3\}, \mathcal{S}_{SR} = \{S_2\}$. The candidates set is sorted by decreasing value of the upper bound, i.e. the first k ones form $\mathcal{U}_k$. Let us consider that k = 2, $C_s(S_1) = C_s(S_2) = 1$ and $C_r(S_2) = C_r(S_3) = 2$. We have $\forall j$, $coef_j = 1$, $max_j = 1$, $min_j = 0$, so $A_j = 1$. The access ratio $r = SB/RB$ = (1/1+1/1)/(1/2+1/4) = 8/3, so a random probe is allowed only after at least r sorted accesses, i.e. 3 sorted accesses before a random one.

- First access is sorted, because $|candidates| < k$. Benefits for S_1 and S_2 computed by **BestSortedSource** are both 0, because N_j = 0 (no top-k candidates yet). Remind that BR-Cost* uses the same **BestSortedSource** as BR-Cost, based on benefit $Bs_j = coef_j \times N_j \times \delta_j / C_s(S_j)$. Source S_1 is then randomly chosen.
- The second access is also sorted ($|candidates| < k$), but this time N_2 = 1 (o_2 not yet read in S_2), while N_1 = 0. Since δ_2 = 1-0.9 = 0.1, we have Bs_2 = 0.1/1 = 0.1, while Bs_1 = 0, so S_2 is chosen.

Access	Retrieved	*candidates*	U_{unseen}
S_1/S	$(o_2, 0.4)$	$\{(o_2, [0.4, 2.4])\}$	2.4
S_2/S	$(o_3, 0.9)$	$\{(o_2, [0.4, 2.3]), (o_3, [0.9, 2.3])\}$	2.3
S_2/S	$(o_1, 0.2)$	$\{(o_3, [0.9, 2.3]), (o_2, [0.4, 1.6]), (o_1, [0.2, 1.6])\}$	1.6
S_3/R	$(o_3, 0.8)$	$\{(o_3, [1.7, 2.1]), (o_2, [0.4, 1.6]), (o_1, [0.2, 1.6])\}$	1.6
S_1/S	$(o_1, 0.3)$	$\{(o_3, [1.7, 2]), (o_2, [0.4, 1.6]), (o_1, [0.5, 1.5])\}$	1.5
...	...	...	...

Fig. 5. First steps of BR-Cost* for k=2 over the example sources in Figure 2

- Now $|candidates|$=k, but *CostCondition* requires a third sorted access before a random probe become possible. We have $N_1 = 1$ (o_3 not yet retrieved in S_1), $N_2 = 1$ (o_2 not yet retrieved in S_2), δ_1 = 0.4-0.3 = 0.1, δ_2 = 0.9-0.2 = 0.7, so Bs_1 = 0.1 and Bs_2 = 0.7, i.e. S_2 is chosen for a sorted access.
- Since *CostCondition* allows now random probes and $U_k = U_{unseen}$ = 1.6, *SortedAccessCondition* returns false and a random access is scheduled. **ChooseCandidate** returns the least refined candidate in $\mathcal{U}_k = \{o_3, o_2\}$. Both objects have been read in one source, but o_3 is returned, because it has a larger interval than o_2. Since $crtmax_2$ = 0.2 and $crtmax_3$ = 1, benefits of random sources are Br_2 = 0.2/2 = 0.1 and Br_3 = 1/2=0.5. Anyway, o_3 has already been read in S_2, so S_3 is the only possible choice for a random probe of o_3.
- *CostCondition* forces again at least three sorted accesses. For the benefit, we have N_1 = 1 (o_3 not yet retrieved in S_1), N_2 = 1 (o_2 not yet retrieved in S_2), δ_1 = 0.4-0.3 = 0.1, δ_2 = 0.2-0.15 = 0.05, so Bs_1 = 0.1 and Bs_2 = 0.05, i.e. S_1 is chosen for a sorted access.
- Execution continues in a similar way until *StopCondition* is satisfied.

3.2 Necessary Choices

As mentioned above, *Necessary Choices* (NC) [19] was the first proposal for a generic algorithm, yet constrained to the case of complete top-k scoring. In this context, NC identifies *necessary accesses* at some moment, as being those for candidates in $\mathcal{U}_k$. Algorithms in the general NC framework do only necessary accesses: each step selects an element in $\mathcal{U}_k$ with incomplete scoring and performs an access for it.

In this framework, NC proposes an algorithm *SR/G* that favors sorted against random accesses for each candidate. SR/G is guided by two parameters: $D = \{d_1, ..., d_m\}$, which indicates *a depth of sorted access* in each S- or SR-source, and H, which indicates *a fixed order of probes* in the random (R and SR) sources for all the candidates. The meaning of D is that sorted access to a source S_j where $crtmax_j \geq d_j$ has always priority against random probes.

Among all the possible pairs (D, H), SR/G selects the optimal one by using sampling optimization. The optimization process converges iteratively: for some initial H, one determines the optimal D, then an optimal H for this D, etc.

Despite its genericity, NC is hardly comparable with BR. In the context of incomplete top-k scoring adopted by BR, NC's analysis of necessary accesses is no longer valid. Source sampling used by SR/G is not always possible and does not guarantee

similar score distribution. We propose here a variant of SR/G, adapted to the context of BR by considering incomplete scoring and a heuristic approximation of (D, H) inspired by BR-Cost*. The intention is to compare *the strategies* proposed by BR-Cost* and SR/G in a context as similar as possible.

The SR/G variant we propose is expressed in the GF framework as follows:

- Besides D and H, a local variable keeps the *best candidate*, i.e. the candidate in $\mathcal{U}_k$ with incomplete scoring having the highest upper bound. SR/G does a first sorted access to some source; the best object is initialized with this first retrieved object and updated after each iteration. Note that at least one object in $\mathcal{U}_k$ has incomplete scoring if the *StopCondition* has not been yet reached.
- *SortedAccessCondition* returns true if the set of sorted sources in which the best candidate has not been yet retrieved and where $crtmax_j \geq d_j$ is not empty.
- **BestSortedSource** returns one of the sources in this set.
- **ChooseCandidate** returns the best candidate.
- **BestRandomSource** returns the first random source not yet probed for the best candidate, following the order defined by H.
- *StopCondition*, for incomplete scoring, is given by (1).

We propose an heuristic approximation of D and H, based on the notion of source benefit used for BR-Cost*.

For H we consider the descending order of the random source benefit computed with (5) and (6).

Estimation of D is based on three hypotheses:

1. The number of sorted accesses to a source must be proportional to the source benefit given by (3).
2. Sorted accesses until depth d_j in each source should produce a decrease of the threshold enough for discriminating the top-k result, which is at least until $U_{unseen} = R_k$, where R_k is the k-th highest real score of an object.
3. If $n_j = N - Nr_j$ is the number of sorted accesses in S_j for reaching depth d_j (see Figure 4), the relation between n_j and d_j depends on the score distribution in sources, generally unknown and approximated here with uniform distribution.

If we note $\Delta_j = max_j - d_j$ the score decrease to reach depth d_j, the three hypotheses above give:

1. $\forall j, n_j = C \times \overline{B_s(S_j)}$, where C is a constant.
2. $U_{max} - R_k = \sum coef_j \times \Delta_j$, where $U_{max} = \mathcal{F}(max_1, ..., max_m)$ is the highest possible aggregated score.
3. $\forall j, n_j/N = (max_j - d_j)/(max_j - min_j)$.

Resolving this equation system produces the following estimation for the depth:

$$d_j = max_j - \frac{A_j^2}{coef_j \times C_s(S_j)} \times \frac{U_{max} - R_k}{\sum_j A_j^2/C_s(S_j)} \tag{8}$$

Real score R_k may be estimated by various methods. This is not important in our experimental evaluation, since we precompute the R_k value, hence considering the best case for SR/G.

Access	Retrieved	*candidates*	U_{unseen}
S_1/S	$(o_2, 0.4)$	$\{(o_2, [0.4, 2.4])\}$	2.4
S_2/S	$(o_3, 0.9)$	$\{(o_2, [0.4, 2.3]), (o_3, [0.9, 2.3])\}$	2.3
S_2/S	$(o_1, 0.2)$	$\{(o_3, [0.9, 2.3]), (o_2, [0.4, 1.6]), (o_1, [0.2, 1.6])\}$	1.6
S_1/S	$(o_1, 0.3)$	$\{(o_3, [0.9, 2.2]), (o_2, [0.4, 1.6]), (o_1, [0.5, 1.5])\}$	1.5
S_1/S	$(o_4, 0.25)$	$\{(o_3, [0.9, 2.15]), (o_2, [0.4, 1.6]), (o_1, [0.5, 1.5]), (o_4, [0.25, 1.45])\}$	1.45
S_1/S	$(o_3, 0.2)$	$\{(o_3, [1.1, 2.1]), (o_2, [0.4, 1.6]), (o_1, [0.5, 1.5]), (o_4, [0.25, 1.45])\}$	1.4
S_3/R	$(o_3, 0.8)$	$\{(o_3, [1.9, 1.9]), (o_2, [0.4, 1.6]), (o_1, [0.5, 1.5]), (o_4, [0.25, 1.45])\}$	1.4
S_2/S	$(o_4, 0.15)$	$\{(o_3, [1.9, 1.9]), (o_2, [0.4, 1.55]), (o_1, [0.5, 1.5]), (o_4, [0.4, 1.4])\}$	1.35
...	...	...	...

Fig. 6. First steps of NC for k=2 over the example sources in Figure 2

Example We take the same example as for BR-Cost*, in the same conditions, to illustrate the first steps of the new variant of NC (see Figure 6).

We first compute $D = \{d_1, d_2\}$ and H. For the two sources with sorted access (S_1 and S_2) we have similar parameters, $max_j = A_j = 1$, $coef_j = 1$, $C_s(S_j) = 1$, $U_{max} = 3$ and $R_k = 1.4$ (the real score of the second best object, which is o_1). Formula 8 gives $d_1 = d_2 = 1 - (3\text{-}1.4)/2 = 0.2$. For H, we have $\overline{B_r(S_3)} = 1/2$ and $\overline{B_{rs}(S_2)} = 1/4$, so the order given by H is $[S_3, S_2]$.

- A first sorted access is done, S_1 is randomly chosen for that.
- o_2 is the best object and among the missing accesses for it, the sorted one in S_2 has priority, because $crtmax_2 = 1 > d_2$, so S_2 is selected for a sorted access.
- o_2 is again the best object and for the same reason as above, S_2 is selected again for a sorted access ($crtmax_2 = 0.9 > d_2$).
- Now o_3 is the best object and among the missing accesses for it, the sorted one in S_1 has priority, because $crtmax_1 = 0.4 > d_1$, so S_1 is selected for a sorted access.
- o_3 is again the best object and for the same reason as above, S_1 is selected again for a sorted access.
- o_3 is still the best object and for the same reason as above, S_1 is selected again for a sorted access.
- o_3 is still the best object, but no more sorted access for it is possible, so the last access for it is considered - the random access for o_3 is scheduled in S_3.
- o_3 being now fully evaluated, o_2 is the new best object. The sorted access for o_2 in S_2 still has priority, because $crtmax_2 = 0.2 \geq d_2$, so S_2 is selected for a sorted access.
- o_2 is again the best object, but the sorted access to S_2 has not priority anymore, because $crtmax_2 = 0.15 < d_2$. The remaining access for o_2 (random probe in S_3) is scheduled. Then execution continues in a similar way until *StopCondition* is satisfied.

3.3 Combined Algorithm

Although Combined Algorithm (CA) [7], limited to SR-sources, is not a generic algorithm, it was a first attempt towards genericity, by proposing to combine NRA and TA strategies to adapt to the case of different costs for random and sorted access.

Access	Retrieved	*candidates*	U_{unseen}
S_1/S	$(o_2, 0.4)$	$\{(o_2, [0.4, 2.4])\}$	2.4
S_1/S	$(o_1, 0.3)$	$\{(o_2, [0.4, 2.4]), (o_1, [0.3, 2.3])\}$	2.3
S_2/S	$(o_3, 0.9)$	$\{(o_2, [0.4, 2.3]), (o_1, [0.3, 2.2]), (o_3, [0.9, 2.2])\}$	2.2
S_2/S	$(o_1, 0.2)$	$\{(o_3, [0.9, 2.2]), (o_2, [0.4, 1.6]), (o_1, [0.5, 1.5])\}$	1.5
S_3/R	$(o_3, 0.8)$	$\{(o_3, [1.7, 2]), (o_2, [0.4, 1.6]), (o_1, [0.5, 1.5])\}$	1.5
...	...	...	...

Fig. 7. First steps of CA-gen for k=2 over the example sources in Figure 2

We propose here *CA-gen*, a generic variant of CA adapted to any source types. Like for CA, if r is the ratio between the average costs of random and sorted access, each sorted (S- and SR-) source is accessed r times, before performing all the random probes for the best candidate in $\mathcal{U}_k$ with *incomplete scoring in random sources*. Note that the best candidate may be different of χ.

Unlike CA, but similar to BR, CA-gen does not produce complete scoring for the best candidate, since its score may still be unknown in some sorted sources. Like for BR, the stop condition corresponds to incomplete top-k scoring.

- The cycle of r sorted accesses in each source can be simulated in GF with local variables indicating the next source to access and the number of accesses already performed in the cycle.
- *SortedAccessCondition* returns true if the cycle is not yet finished.
- **BestSortedSource** simply returns the next source.
- **ChooseCandidate** returns the best candidate, as defined above.
- **BestRandomSource** returns the first random source not yet probed for the best candidate. If no such source exists, the cycle stops.
- *StopCondition*, for incomplete scoring, is given by (1).

Example We take the same example as for BR-Cost* and NC, in the same conditions, to illustrate the first steps of CA-gen (see Figure 7).

The access ratio $r = C_r(S_j)/C_s(S_j) = 2$, so two sorted accesses are scheduled in each source before fully evaluating the best object.

- $r = 2$ sorted accesses to S_1, then to S_2 are done.
- o_3 is the best object and we schedule all the random probes for it. The only possible one is the access to S_3, because o_3 has been already read in S_2 through a previous sorted access. Note that o_3 is not fully evaluated after this step because its value in S_1 is missing and cannot be obtained by random access.
- Execution continues in the same way, by cycles of two sorted accesses in each source followed by random probes for the best object, until *StopCondition* is satisfied.

4 Approximation by early stopping

Top-k processing in our context is usually expensive because of predicate evaluation, therefore reducing the execution cost by accepting approximate results is a promising

approach. We adopt the method proposed by TA$_\theta$ [7], based on relaxing the threshold condition in TA with a factor $\theta > 1$, i.e. the algorithm stops when the score of at least k candidates exceeds U_{unseen}/θ. This produces a θ-approximation of the top-k result, i.e. a set K_a of k objects such that $\forall x \in K_a, \forall y \notin K_a, \theta \times score(x) \geq score(y)$. As explained in the related work section, a θ-approximation guarantees that the ratio between the score of the best missed object in the approximate result (best false negative) and that of the worst false positive cannot exceed θ.

Note that this method is equivalent to an *early stopping* of the exact algorithm, i.e. TA and TA$_\theta$ have the same execution until the end of TA$_\theta$, which occurs first.

We generalize here the TA$_\theta$ approach to the case of incomplete scoring within the GF framework, i.e. to any top-k algorithm in our context, and thus enable the comparison of the behavior of top-k algorithms in the case of approximate results by early stopping.

Note that TA$_\theta$ considers that all source scores belong to the $[0, 1]$ interval. In the general case, in order to preserve the meaning of θ-approximations, we simply consider that scores in source S_j belong to a $[0, max_j]$ interval.

Consider an approximate solution K_a composed of k candidates with possibly incomplete scoring at some point during the execution of the algorithm in the GF framework. Then the condition for detecting K_a as being surely a θ-approximation of the top-k result is given by the following theorem.

Theorem 1. *An approximate solution K_a composed of k candidates with incomplete scoring during the execution of a top-k algorithm is surely a θ-approximation of the top-k result* ***iff***

$$\theta \times min_{c \in K_a}(L(c)) \geq max_{c \notin K_a}(U(c)) \tag{9}$$

Proof. At the given moment during the execution, $min_{c \in K_a}(L(c))$ represents the minimum possible score for a candidate in K_a, while $max_{c \notin K_a}(U(c))$ is the maximum possible score for an object not in K_a (including unseen objects).

We first show that if condition (9) is true, then K_a is a θ-approximation of the exact result.

For any candidate c, we have $L(c) \leq score(c) \leq U(c)$. More generally, for any unseen object o, we have $U(o) = U_{unseen}$, its maximum possible score. Then $\forall x \in K_a, score(x) \geq L(x) \geq min_{c \in K_a}(L(c))$ and $\forall y \notin K_a, max_{c \notin K_a}(U(c)) \geq U(y) \geq score(y)$. If the theorem condition holds, then $\forall x \in K_a, y \notin K_a, \theta \times score(x) \geq score(y)$, i.e. K_a is a θ-approximation.

We demonstrate the reverse implication by using proof by contradiction: if condition (9) is not true, then K_a is not surely a θ-approximation.

Consider now $x = argmin_{c \in K_a}(L(c))$ the candidate with the worst minimal score in K_a and $y = argmax_{c \notin K_a}(U(c))$ the object with the best maximal score outside of K_a. If the theorem condition does not hold for K_a, then $\theta \times L(x) < U(y)$, so it is possible that $\theta \times score(x) < score(y)$, i.e. K_a may not be a θ-approximation.

In the GF context, algorithms manage only the set of candidates discovered in sorted sources. Considering $K_a \subset candidates$, the stop condition (1) becomes:

$$\theta \times min_{c \in K_a}(L(c)) \geq max(U_{unseen}, max_{c \in candidates - K_a}(U(c))) \tag{10}$$

The difference with Theorem 1 is that here U_{unseen} gives the upper bound score for all the objects not yet discovered and thus not members of *candidates*.

Theorem 2. *Eliminating non-viable candidates does not affect the stop condition (10).*

Proof. Suppose that at some moment a non viable candidate x affects the stop condition. Since $x \notin K_a$, x can only impact the right side of the inequality and only if $U(x) > U_{unseen}$ and $U(x) > U(y), \forall y \in candidates - K_a$. But $U(x) < L_k$ (x non-viable), so all the objects in $candidates - K_a$ are non-viable and $L_k > U_{unseen}$, which in accordance to (1) means that at the current moment the exact top-k has been already found, i.e. the algorithm is already stopped.

To estimate the precision of an approximate solution, we propose a distance measure based on two principles:

- Order of elements in the top-k solution is not important.
- Only wrong elements (false positives) in the approximate solution affect precision, i.e. the quality of the approximate result is given by the quality of the false positives.

Distance is measured by the difference between the real scores of wrong elements and R_k, the k-th score in the exact solution, normalized to the $[0, 1]$ interval by dividing it by R_k. Indeed, R_k is the maximum possible distance to R_k, since the lowest possible global score is 0.

The distance between an element $o \in K_a$ and the real top-k result K is defined as follows:

$$\text{dist}(o, K) = \begin{cases} (R_k - score(o))/R_k, & \text{if } o \notin K \\ 0, & \text{if } o \in K \end{cases} \tag{11}$$

The global distance between an approximate solution K_a and K is defined as the average of the individual distances between elements of K_a and K.

$$\text{dist}(K_a, K) = \frac{1}{k} \sum_{o \in K_a} \text{dist}(o, K) \tag{12}$$

We measure *the quality* of an approximate solution K_a as being $1 - dist(K_a, K)$.

The relation between our distance measure and θ-approximations is given by the following theorem.

Theorem 3. *If K_a is a θ-approximation of the real solution K, then $dist(K_a, K) \leq \theta - 1$. Moreover, the $\theta - 1$ value is optimal in the general case, i.e. it is the smallest upper bound of $dist(K_a, K)$ that can be guaranteed.*

Proof. If $K_a = K$ then $dist(K_a, K) = 0$ and the inequality is true. Otherwise, considering $x \in K - K_a$, then $score(x) \geq R_k$. K_a being a θ-approximation of K, $\forall o \in K_a, \theta \times score(o) \geq score(x) \geq R_k$, so $R_k - score(o) \leq (\theta - 1) score(o)$.

In conclusion, $dist(K_a, K) = \frac{1}{k} \sum_{o \in K_a} dist(o, K) = \frac{1}{k} \sum_{o \in K_a - K} \frac{R_k - score(o)}{R_k} \leq \frac{1}{k} \sum_{o \in K_a - K} \frac{(\theta - 1) score(o)}{R_k} = \frac{\theta - 1}{k R_k} \sum_{o \in K_a - K} score(o) \leq \frac{\theta - 1}{k R_k} k R_k = \theta - 1$.

Moreover, no distance smaller than $\theta - 1$ can be guaranteed. Indeed, in the general case it is possible for K to be composed of k objects of score s, while the approximate solution K_a may be a set of k objects of score s/θ. Given the definition, this possible K_a is a θ-approximation of K, with $dist(K_a, K) = \theta - 1$.

We propose in this paper a comparative study of the approximation potential of multi-criteria top-k algorithms.

We draw cost-distance curves for these algorithms and compare their shapes. A point on the cost-distance curve indicates the quality of the approximate result on early stopping at that moment/cost. Since arbitrary early stopping comes with no guarantees on the precision of the approximate result, we also produce θ-approximations in each case and compare costs for measured and guaranteed precision.

5 Experiments

We experimentally compare the BR strategy with that of the other generic algorithms in terms of *execution cost*. Then, we compare the *approximation potential* of various categories of state-of-the-art top-k algorithms, both generic and specific.

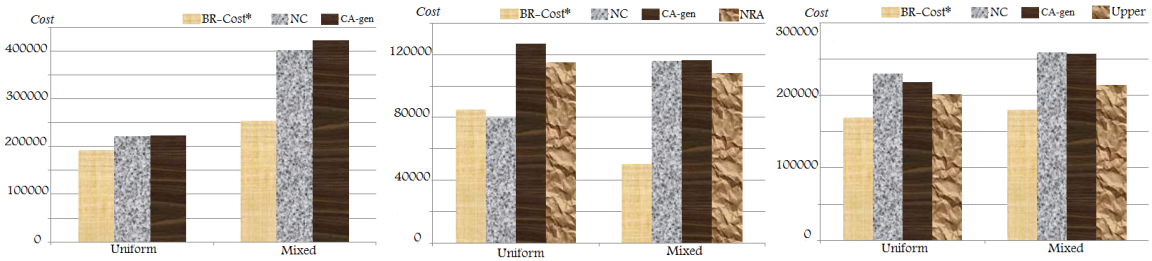
Data sets and parameters

We use synthetic sources, independently generated as lists of scores in the $[0, 1]$ interval for the N objects, then organized for S, R or SR access, depending on the source type. We consider two types of score distribution in a source: uniform or exponential ($p(x) = \lambda e^{-\lambda x}$), for $\lambda = 1$ and restricted to the $[0, 1]$ interval. Exponential distribution illustrates S-sources where scores have fast decrease at the beginning, potentially more discriminant than sources with uniform distribution. The choice of synthetic data is motivated by the need for an experimental testbed with a relatively high number of criteria (up to 18 in our tests), which was not provided by the real data sets we could find.

We measure the execution costs for each algorithm as the sum of costs of all the source accesses for computing the top-k result. We consider that all the sorted (random) accesses have the same cost C_s (C_r). Each result in the experiments is the average of 10 measures over different randomly generated sources. We consider weighted sum as the aggregation function, with coefficients randomly generated for each of the 10 measures.

The following parameters are considered in the experiments:

- The number of database objects is $N = 10\ 000$.
- Queries are looking for the best $k = 50$ objects.
- We consider 6 S-sources, 6 R-sources and 6 SR-sources.
- We consider the most common cost settings, with random accesses more expensive than sorted ones: C_r=10, C_s=1.
- Two configurations for data distribution in sources are considered: *uniform* for all the sources or *mixed*, i.e. exponential distribution for half of the sorted sources (3 S-sources and 3 SR-sources), uniform for the other sources.



(a) All source types (b) No R-sources (c) No S-sources

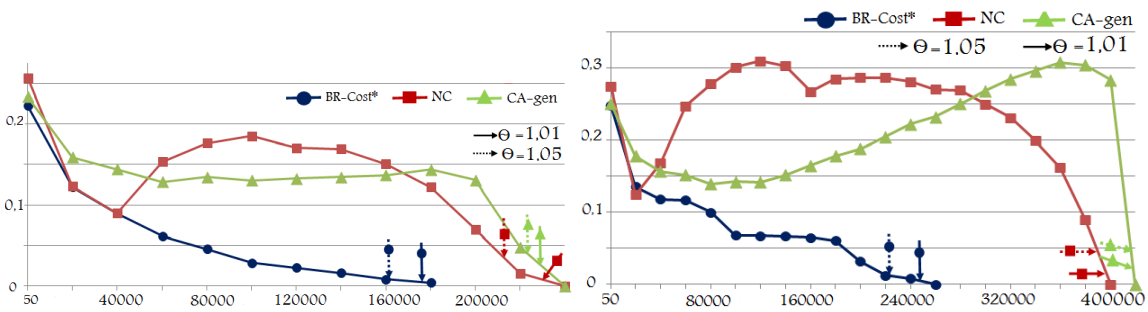
Fig. 8. Execution cost comparison

Comparison of the execution cost

We compare the execution cost of BR-Cost* with the NC variant and CA-gen in three configurations of source types: no R-sources, no S-sources, and all the source types. We also add to the comparison the reference non-generic algorithms compatible with that setting. In each configuration, uniform and mixed data distribution are considered.

- *All source types* (S, R and SR).
  Figure 8.a shows that BR-Cost* behaves visibly better (10%) than both NC and CA-gen for uniform distribution, while the difference becomes important for mixed distribution: approximately 37% better than NC and 40% better than CA-gen.
- *No R-sources* (only S and SR).
  Note that here cost and source settings are in favor of algorithms that realize only sorted access (NRA) or strongly favors them (NC). Figure 8.b shows that in the uniform distribution case BR-Cost* and NC are the best, with very close costs, much better than CA-gen (around 33%), which is even outperformed by NRA. For mixed distribution, BR-Cost* is clearly much better than NC and CA-gen (almost 60%), which are outperformed by NRA.
- *No S-sources* (only R and SR).
  Figure 8.c shows that in all the cases BR-Cost* outperforms the other algorithms and that NC and CA-gen are less adapted to this setting, performing worse than Upper. The benefit of using BR-Cost* is bigger in the mixed distribution case (around 28% better than NC and CA-gen) compared to uniform distribution (24%). Compared to Upper, the benefit is similar in both cases, around 15%.

In conclusion, BR-Cost* successfully adapts to various source types and data distribution settings, and outperforms not only the other generic approaches, but also specific algorithms designed for that case. We also note a weakness for the other generic strategies in one of the studied cases: no S-sources for NC and no R-sources for CA-gen. Paradoxically, mixed distribution does not improve cost in most cases; we guess that discriminant distributions are counter-balanced here by the lack of correlation between sources and by their relatively high number.



(a) All source types, uniform distribution

(b) All source types, mixed distribution

Fig. 9. Approximation with $\mathcal{U}_k$, all source types

Approximation potential

We measure the potential of approximation by early stopping of the top-k algorithms by drawing their cost-distance curves.

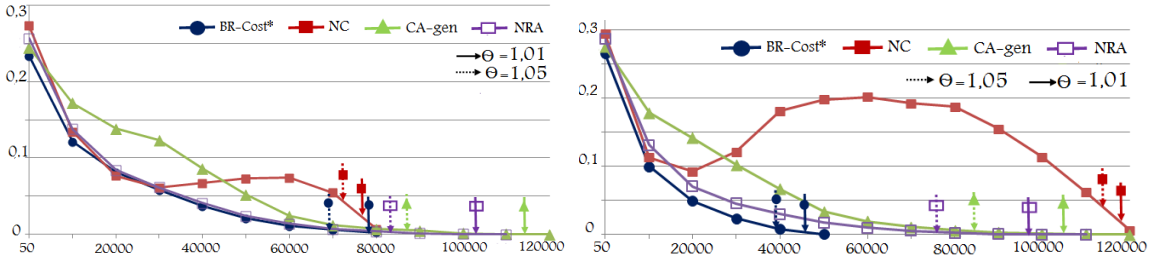
Distance between approximate and real solution is computed with formulas (11)-(12). We measure this distance in several points during the algorithm's normal execution, every 2000 cost units (or every 1000 for the no R-source case, where cost is smaller), then we extrapolate a curve relying these points. Each point on the curve represents the distance between the approximate solution and the real one if the algorithm stops at that moment. A curve "below" another one indicates a better approximation potential.

The form of the curve also indicates *approximation stability*. A monotone descending curve means stable approximation, that improves if execution continues, while non-monotony indicates an algorithm badly adapted for approximation by early stopping.

For each cost-distance curve we measure the end point that corresponds to a θ-approximation obtained with the stop condition (10). We consider two values, $\theta = 1.05$ and $\theta = 1.01$, that correspond to a guaranteed distance of 0.05, respectively 0.01 (see Theorem 3). We compare the position of these points with that of the intersection between the curve and the corresponding distance.

We consider two cases for the approximate solution. The first one is the set $\mathcal{U}_k$ of k candidates with the highest upper bound. This is a natural choice for the approximate solution, since $\mathcal{U}_k$ is the set of candidates that top-k algorithm focus on during execution. More precisely, all the algorithms proposed so far base their strategies on $\mathcal{U}_k$, either for deciding a sorted access, or for the choice of a candidate for random probes. Intuitively, candidates with high upper bounds must be "refined" because their upper bound make them potentially belong to the final top-k. The algorithm *must* decide if they really belong to the result or not - if not, the algorithm cannot end without refining the candidate's score to make it non-viable.

The second proposal for an approximate solution is the set of k candidates with the highest lower bound $\mathcal{L}_k$. Intuitively, belonging to $\mathcal{L}_k$ means that the candidate was already refined with good scores in some sources. This may be a good indication that



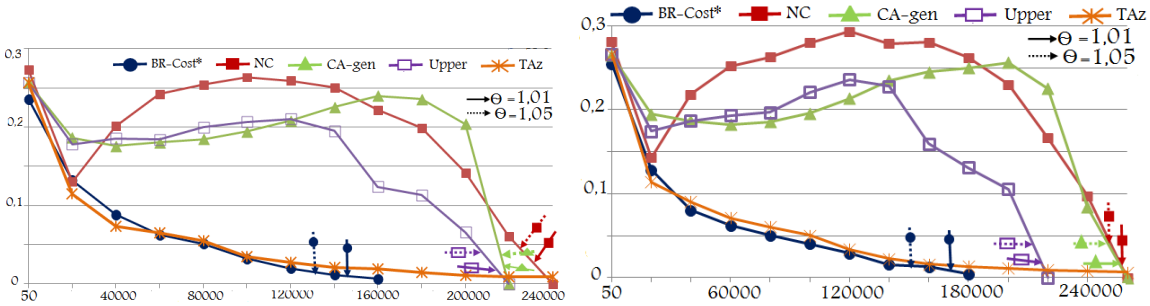
(a) No R-sources, uniform distribution (b) No R-sources, mixed distribution

Fig. 10. Approximation with $\mathcal{U}_k$, no R-sources

the candidate belongs to the final top-k, better than for $\mathcal{U}_k$ where high upper bounds may be the result of little refinement, thus with high uncertainty.

Approximation with $\mathcal{U}_k$ Figures 9-11 present the cost-distance curves for uniform and mixed data distributions in the three cases of source types. Final costs for algorithms may be less visible in these figures, but they can also be retrieved in Figure 8.

- *All source types*.
  We compare the generic algorithms BR-Cost*, the NC variant and CA-gen.
  For uniform distribution (Figure 9.a), BR-Cost* approximation distance quickly decreases and the algorithm has clearly better approximation properties than CA-gen (much higher distance, only decreasing at the end) or NC (totally unstable).
  Mixed distribution (Figure 9.b) accentuates the problems of NC and CA-gen (which becomes unstable), while BR-Cost* keeps a good curve shape. However, θ-approximation significantly reduces the cost saving for BR-Cost*, e.g. for θ=1.05 algorithm stops at cost 160 000, while the corresponding distance of 0.05 is already reached at cost 70 000.
- *No R-sources* (only S and SR).
  Besides the three generic algorithms, we also consider here the NRA algorithm.
  Excepting NC, algorithms produce in this case stable approximations. For uniform distribution (Figure 10.a), BR-Cost* and NRA have very close curves, i.e. close approximation potential, but BR-Cost* produces θ-approximations with better costs. Similarly, CA-gen has good approximation potential, especially in the second half of the execution, but θ-approximations are more expensive than for NRA. NC is more stable than in the previous case and and its low execution time helps it producing less expensive $\theta-$approximations.
  For mixed distribution (Figure 10.b), BR-Cost* improves its potential compared to NRA, while NC becomes highly unstable.
- *No S-sources* (only R and SR).
  Besides the three generic algorithms, we also study here the Upper and TAz algorithms. Despite the fact that it is much more expensive (around six times the cost of BR-Cost*), TAz is considered because of the good approximation potential of algorithms with many SR-sources, which reduce as much as possible the uncertainty of the candidates' scores.



(a) No S-sources, uniform distribution

(b) No S-sources, mixed distribution

Fig. 11. Approximation with $\mathcal{U}_k$, no S-sources

For both uniform (Figure 11.a) and mixed distribution (Figure 11.b), behavior is very similar. CA-gen, NC and Upper are highly unstable, while BR-Cost* and TAz have very close curves, with very good approximation potential. However, because of its high execution cost, θ-approximations of TAz are much more expensive than for BR-Cost* (because of their high values, final cost and θ-approximations for TAz are not visible in the figure).

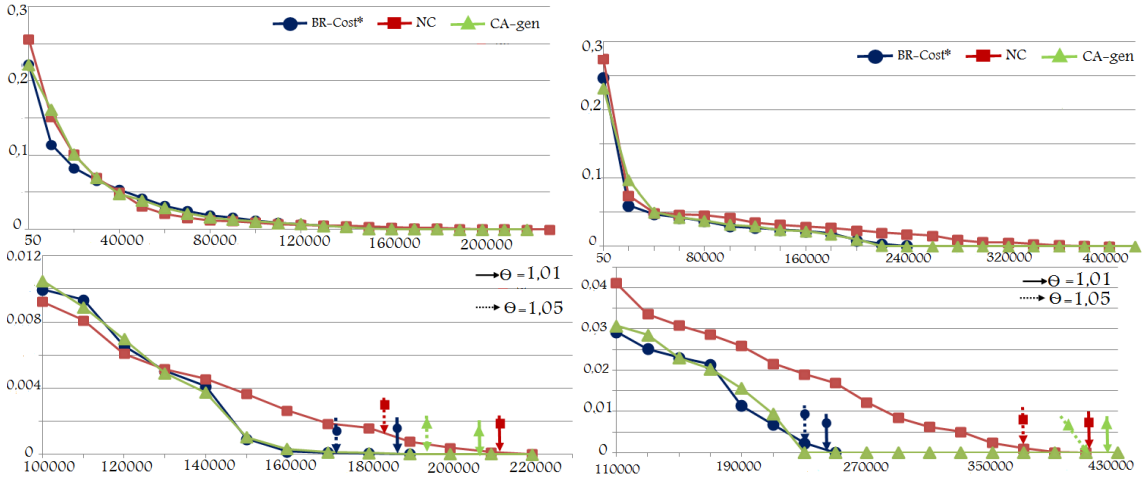
In conclusion, BR-Cost* has clearly the best approximation potential with $\mathcal{U}_k$ among the generic algorithms, with good properties for the different data distributions. The other generic algorithms are badly adapted to approximation with $\mathcal{U}_k$: NC and CA-gen are systematically unstable.

We guess that the good approximation properties of BR-Cost* come from its breadth-first strategy. Handling the current top-k $\mathcal{U}_k$ as a whole, instead of focusing on the best candidate only, produces a more stable evolution of $\mathcal{U}_k$ toward the final solution.

The price to pay for guaranteed precision in θ-approximations is important for algorithms with good approximation curves - we notice a significant difference with the potential cost for the same approximation quality. The cost of θ-approximations appears to be dependent on the total cost of the algorithm: for algorithms with very close cost-distance curves, higher total costs systematically lead to higher θ-approximation costs.

Approximation with $\mathcal{L}_k$ Figures 12-14 present the cost-distance curves for the approximation with the best k lower bound scores $\mathcal{L}_k$, in the three cases of source types. The sub-figure for each case presents, besides the curves, a zoom on the final part of the execution, where the curves are very close.

– *All source types.*
For both uniform (Figure 12.a) and mixed distribution (Figure 12.b), the curves for all the algorithms are very close, stable, with good approximation potential.
BR-Cost* and CA-gen have slightly better curves than NC, the difference being visible in the mixed distribution case and on the final part of the uniform case.



(a) All source types, uniform distribution (b) All source types, mixed distribution

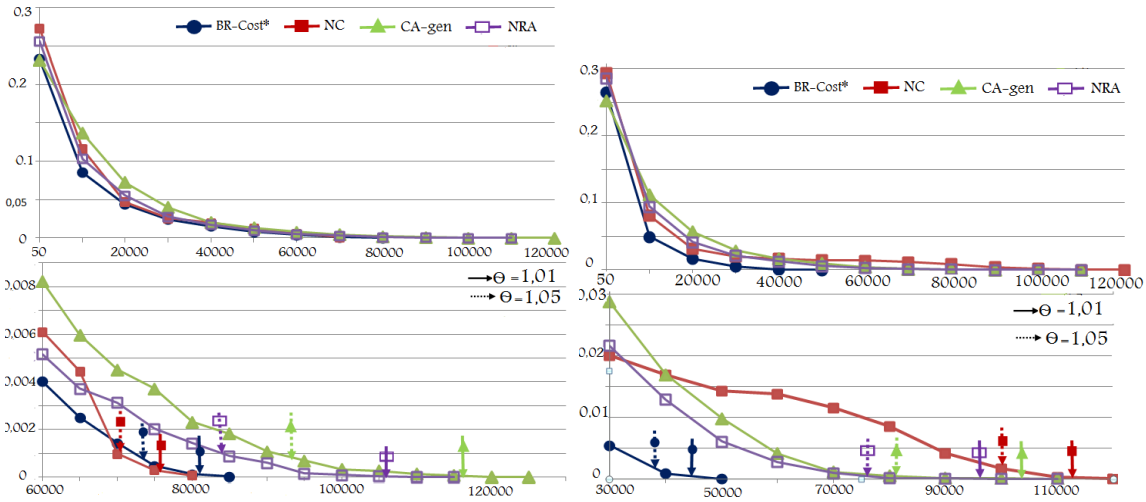
Fig. 12. Approximation with $\mathcal{L}_k$, all source types

Comparison of θ-approximations follows the conclusion of the previous point, algorithms with better execution costs produce better θ-approximations, i.e. BR-Cost* is the best, while CA-gen and NC are very close.
We notice that cost-distance curves with $\mathcal{L}_k$ are better than those with $\mathcal{U}_k$ in all the cases. This also leads to an increased difference between the cost with θ-approximation and the potential one.

– *No R-sources*.
Conclusions are similar to the all source types case for both curve shapes and θ-approximations, but differences between algorithms are more important here.
For uniform distribution (Figure 13.a), BR-Cost* has globally the best shape, followed very closely by NC and NRA, while CA-gen is slightly, but visibly worse.
For mixed distribution (Figure 13.b), the superiority of BR-Cost* is clearer, the other algorithms being close and having sections on which they have better approximation potential than the others.

– *No S-sources*.
We find similar conclusions in this case too, with the remark that generic algorithms have globally better curve shapes than Upper and TAz.
For uniform distribution (Figure 14.a), BR-Cost*, CA-gen and NC are very close, with CA-gen slightly better on the middle part and NC slightly worse on the final part. Upper has globally the least favorable approximation potential, TAz becoming worse at the end only because of its higher cost.
For mixed distribution (Figure 14.b), BR-Cost* and CA-gen have clearly the best potential, followed by NC. Unlike for uniform distribution, Upper is here globally better than TAz.

In conclusion, we notice that approximation with $\mathcal{L}_k$ has better quality than with $\mathcal{U}_k$ for all the algorithms. Compared with the $\mathcal{U}_k$ case, approximation is always stable with



(a) No R-sources, uniform distribution (b) No R-sources, mixed distribution

Fig. 13. Approximation with $\mathcal{L}_k$, no R-sources

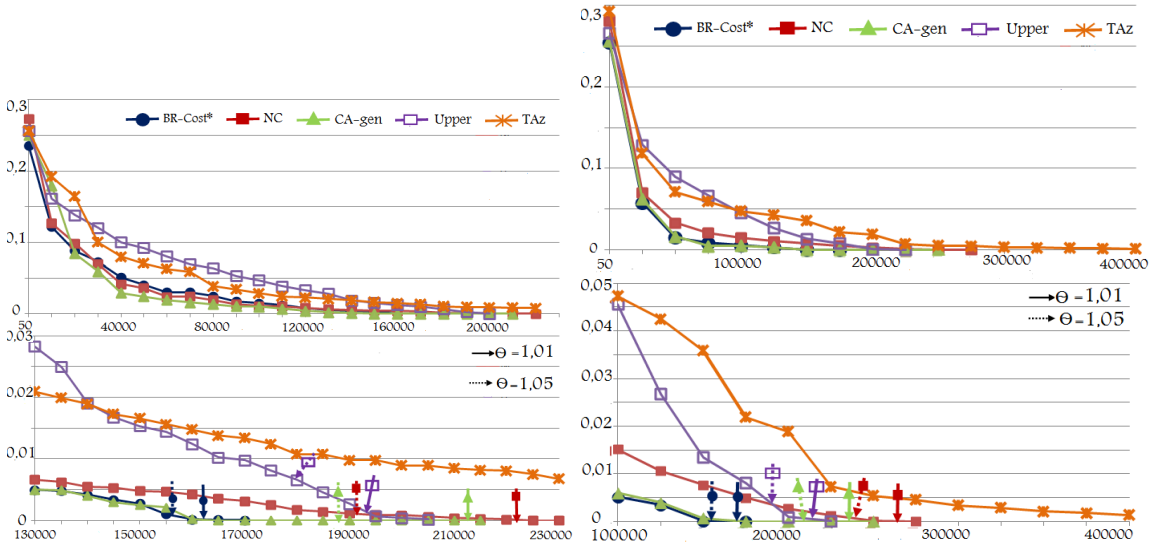
$\mathcal{L}_k$ and has better precision at the same execution cost. Even if BR-Cost* has globally the best properties, the approximation potential of generic algorithms is very close in this case.

However, θ-approximations are not improved by $\mathcal{L}_k$ and lead to an increased difference between the potential cost and that for guaranteed precision.

6 Conclusion

In this paper we proposed a generic framework GF for top-k processing over expensive ranking predicates, able to express any top-k algorithm. We compared within this framework our generic algorithm BR with generic variants that we proposed for algorithms NC and CA, adapted to a similar context. Comparison of the algorithm strategies within GF was completed with experimental measures indicating that the breadth-first strategy of BR adapts itself very well to various source type configurations and data distributions, leading to better execution cost than the other generic or specific strategies.

We also presented a study of the approximation potential of top-k algorithms by early stopping, by proposing a generalization of θ-approximation in the context of the GF framework and an experimental comparison between algorithms for two common approximation sets: candidates with best k upper bounds ($\mathcal{U}_k$) and with best k lower bounds ($\mathcal{L}_k$). By comparing cost-distance curves we concluded that the BR strategy globally has the best approximation potential, with a clear advantage on the others in the $\mathcal{U}_k$ approximation case. However, $\mathcal{L}_k$ approximation produces better results for all the algorithms and greatly reduces the differences between them. We noticed that θ-approximation is weakly correlated with the approximation potential and significantly



(a) No S-sources, uniform distribution (b) No S-sources, mixed distribution

Fig. 14. Approximation with $\mathcal{L}_k$, no S-sources

depends on the total execution cost. This cancels the difference between $\mathcal{U}_k$ and $\mathcal{L}_k$ approximation and favors again the BR strategy, which produces better total costs.

References

1. R. Akbarinia, E. Pacitti, and P. Valduriez. Best position algorithms for top-k queries. In VLDB, pages 495–506, 2007.
2. M. Badr and D. Vodislav. A general top-k algorithm for web data sources. In DEXA, pages 379–393, 2011.
3. C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Comput. Surv., 33(3):322–373, Sept. 2001.
4. N. Bruno, L. Gravano, and A. Marian. Evaluating top-k queries over web-accessible databases. In ICDE, pages 369–, 2002.
5. P. Cao and Z. Wang. Efficient top-k query calculation in distributed networks. In PODC, pages 206–215, 2004.
6. K. C.-C. Chang and S. won Hwang. Minimal probing: supporting expensive predicates for top-k queries. In SIGMOD Conference, pages 346–357, 2002.
7. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci., 66(4):614–656, 2003.
8. U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing multi-feature queries for image databases. In VLDB, pages 419–428, 2000.
9. U. Güntzer, W.-T. Balke, and W. Kießling. Towards efficient multi-feature queries in heterogeneous environments. In ITCC, pages 622–628, 2001.
10. I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. VLDB J., 13(3):207–221, 2004.

11. I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-*k* query processing techniques in relational database systems. ACM Comput. Surv., 40(4), 2008.
12. C. Li, K. C.-C. Chang, and I. F. Ilyas. Supporting ad-hoc ranking aggregates. In SIGMOD Conference, pages 61–72, 2006.
13. C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. Ranksql: Query algebra and optimization for relational top-k queries. In SIGMOD Conference, pages 131–142, 2005.
14. N. Mamoulis, K. H. Cheng, M. L. Yiu, and D. W. Cheung. Efficient aggregation of ranked inputs. In ICDE, page 72, 2006.
15. A. Marian, N. Bruno, and L. Gravano. Evaluating top-*k* queries over web-accessible databases. ACM Trans. Database Syst., 29(2):319–362, 2004.
16. S. Michel, P. Triantafillou, and G. Weikum. Klee: A framework for distributed top-k query algorithms. In VLDB, pages 637–648, 2005.
17. A. Natsev, Y.-C. Chang, J. R. Smith, C.-S. Li, and J. S. Vitter. Supporting incremental join queries on ranked inputs. In VLDB, pages 281–290, 2001.
18. M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In VLDB, pages 648–659, 2004.
19. S. won Hwang and K. C.-C. Chang. Optimizing top-k queries for middleware access: A unified cost-based approach. ACM Trans. Database Syst., 32(1):5, 2007.