Generic Top-k Query Processing with Breadth-First Strategies

Mehdi Badr and Dan Vodislav

ETIS, ENSEA, University of Cergy-Pontoise, CNRS, France
`firstname.lastname@u-cergy.fr`

Abstract. Many algorithms for top-k query processing with ranking predicates have been proposed, but little effort has been directed toward genericity, i.e. supporting any type (sorted and/or random) or cost settings for the access to the lists of predicate scores. 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, then we compared it with specific top-k algorithms. In this paper, we compare the BR breadth-first strategy with other existing generic strategies and experimentally show that BR leads to better execution costs. 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. 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 procesing query, ranking, multicriteria information retrieval.

1 Introduction and Related Work

We address the problem of top-k 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 criteria. 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 predicates. 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.

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 from a touristic application, where the visitor of a monument takes a

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 top-k query

photo of a detail and searches for the "best" monument on three criteria: *near* to its current location, reproducing a *similar* detail, and *exposing Renaissance sculptures*.

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 search in highly dimensional spaces, based often on specific multidimensional index structures [3].

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. In the 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 the access to 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 next highest score s and the corresponding object identifier *o*.
- *R-source*: random access only, through the operator *getScore(S, o)* returning the score of 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 scores for each of them. 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 id for R-sources. Local scores belong to the $[0, 1]$ interval, 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). 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

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	$(o_1, 0.2)$	$\{(o_2, [0.5, 1.5]), (\underline{o_3, [1.7, 2.1]}), (o_1, [0.2, 1.6])\}$	1.6

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

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, etc. Execution ends when the minimum global score of o_3 exceeds both U_{unseen} and the maximum global score of all the other candidates, i.e. o_3 is the best (top-1) object.

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 disposing of statistical information about scores, or disposing also of 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 Figure 2 and propose their own heuristic for deciding the next access. 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}. *QuickCombine*[8], uses the same idea as StreamCombine to select the next sorted source to probe. *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. *NC*[19] (Necessary Choices) proposes a framework for generic top-k algorithms in the case of results *with complete scoring*, a strategy SR that favors sorted accesses, and a specific algorithm SR/G that uses sampling optimization to find the best fit with the source settings.

Approximate top-k processing has been considered in several approaches. 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 of k objects such that $\forall x \in K$, $\forall y \notin K, \theta score(x) \geq score(y)$ (global and local scores are considered to belong to the [0,1] interval). 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. 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.

This paper completes the BR approach with the following contributions:

- A general framework *GF* for generic top-k processing, that allows expressing any top-k algorithm in our context, thus providing a basis for comparative analysis.
- New, comparable generic variants of the BR, NC and CA algorithms with experimental comparison, showing that the BR strategy leads to better costs.
- A generalization of θ-approximation computing in the context of GF, and a first experimental study of the ability of generic top-k algorithms to produce good approximate results on early stopping, showing that the BR strategy has a better approximation potential.

The rest of the paper is organized as follows: the next section introduces the generic framework for top-k processing and compares in this context the BR algorithm with generic variants of NC and CA; Section 3 presents our approach for top-k approximation, then we report experimental results and end with conclusions.

2 Generic Top-k Framework and Algorithms

We propose *GF*, a *generic framework* for top-k processing (Figure 3). 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 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 $\mathcal{U}_k(\mathcal{L}_k)$ the current subset of k candidates with the best k upper (lower) bound scores[1]. 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. We note $\chi \in \mathcal{U}_1$ the candidate with the current best upper bound score.

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 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

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

i.e. only k candidates are viable and there is no viable unseen object. It is simple to demonstrate that this condition is necessary and sufficient for a correct top-k result. 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 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.

Given its ability to express any top-k algorithm, the GF framework is a valuable tool for comparing top-k strategies. In the 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.

2.1 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

[1] 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

top-k candidates $\mathcal{U}_k$ as a whole, instead of focusing on the best candidate χ, which is the common approach.

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

The *SortedAccessCondition* in the BR strategy combines three conditions: $|candidates| < k$ **or** $U_{unseen} > U_k$ **or** *CostCondition()*. 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: it maintains the whole current top-k free of unseen objects, while the common strategy is to consider only the best candidate ($U_{unseen} > U(\chi)$).

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. 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 best one was *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** adopts the benefit-oriented strategy proposed by StreamCombine [9] for choosing a sorted source. The benefit of source S_j is $B_j = (\partial\mathcal{F}/\partial S_j) \times N_j \times \delta_j / C_s(S_j)$, 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*

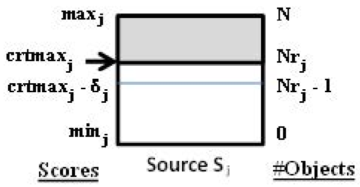


Fig. 4. Scores in a sorted source S_j

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.

BestRandomSource uses also a benefit-oriented strategy inspired from algorithms with controlled random probes such as Upper [4]; the benefit of source S_j is $B_j = coef_j \times (crtmax_j - min_j)/C_r(S_j)$, 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.

We propose here *BR-Cost**, an improved variant of BR-Cost, using a different method for estimating r as a *ratio of benefits*. Roughly speaking, the benefit of an access to a source is the ratio between the refinement produced on the candidate score intervals and the cost of that access. This approach 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:

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

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 $Nr_j \approx N/2$:

$$\overline{B_s}(S_j) \approx coef_j \times (max_j - min_j)/C_s(S_j) \tag{2}$$

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:

$$\begin{aligned} B_{rs}(S_j) &= coef_j \times (crtmax_j - min_j)/C_r(S_j) \\ \overline{B_{rs}}(S_j) &\approx coef_j \times (max_j - min_j)/2C_r(S_j) \end{aligned} \tag{3}$$

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

$$B_r(S_j) = \overline{B_r}(S_j) = coef_j \times (max_j - min_j)/C_r(S_j) \tag{4}$$

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.

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

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

In conclusion, the access ratio r used by BR-Cost* is:

$$r = SB/RB = \frac{\sum_{S_j \in \mathcal{S}_S \cup \mathcal{S}_{SR}} \frac{A_j}{C_s(S_j)}}{\sum_{S_j \in \mathcal{S}_R} \frac{A_j}{C_r(S_j)} + \sum_{S_j \in \mathcal{S}_{SR}} \frac{A_j}{2C_r(S_j)}} \tag{5}$$

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

2.2 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 $\mathcal{U}_k$.

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 couples (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 realizes 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 (3) and (4). 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 (2).
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, we obtain:

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:

$$d_j = max_j - \frac{A_j^2}{coef_j \times C_s(S_j)} \times \frac{U_{max} - R_k}{\sum_j A_j^2/C_s(S_j)} \tag{6}$$

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.

2.3 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.

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.

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. Then *SortedAccessCondition* returns true if the cycle is not yet finished and **BestSortedSource** simply returns the next source. **ChooseCandidate** returns the best candidate, as defined above and **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).

3 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 $\forall x \in K_a, \forall y \notin K_a, \theta \times score(x) \geq score(y)$.

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 in the case of incomplete scoring within the GF framework and thus enable a comparison between various top-k algorithms.

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 a θ-approximation of the top-k result is given by the following theorem.

Theorem 1. *An approximate solution K_a composed of k objects with incomplete scoring is surely a θ-approximation of the top-k result* ***iff*** $\theta \times min_{c \in K_a}(L(c)) \geq max_{c \notin K_a}(U(c))$

Proof. Since $L(c) \leq score(c) \leq U(c)$, 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.

Consider now $x = argmin_{c \in K_a}(L(c))$ and $y = argmax_{c \notin K_a}(U(c))$. 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:

$$\theta \times min_{c \in K_a}(L(c)) \geq max(U_{unseen}, max_{c \in candidates - K_a}(U(c))) \tag{7}$$

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 (7).*

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 the exact top-k has been already found.

To estimate the precision of an approximate solution, we propose a distance measure based on two principles: (i) order and final scores of elements in the top-k solution are not important, and (ii) only wrong elements in the approximate solution affect precision.

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 result K, respectively between K_a and K are defined as follows:

$$\text{dist}(o, K) = \begin{cases} \frac{R_k - score(o)}{R_k}, & \text{if } o \notin K \\ 0, & \text{if } o \in K \end{cases}, \quad \text{dist}(K_a, K) = \frac{1}{k} \sum_{o \in K_a} \text{dist}(o, K) \quad (8)$$

We measure *the precision* 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 $dist(K_a, K) \leq \theta - 1$.*

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

We propose here a comparative study of the approximation potential of generic top-k algorithms. We draw cost-distance curves for these algorithms and compare their shapes. A point on the cost-distance curve indicates the precision 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.

4 Experiments

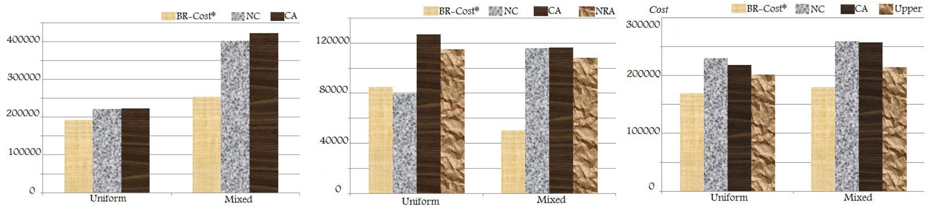
We experimentally compare the BR strategy with that of the other generic algorithms in terms of *execution cost* and of *approximation potential.*

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.

We measure the execution costs for each algorithm as the sum of costs of all the source accesses. 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: $N = 10000$ database objects, $k = 50$, 6 sources of each type, and 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.



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

Fig. 5. Execution cost comparison

exponential distribution for half of the sorted sources (3 S-sources and 3 SR-sources) and uniform for the other sources.

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, all 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 (Figure 5.a). BR-Cost* behaves visibly better (10%) than both NC and CA-gen for uniform distribution, while the difference becomes important for mixed distribution: approximatively 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 5.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.

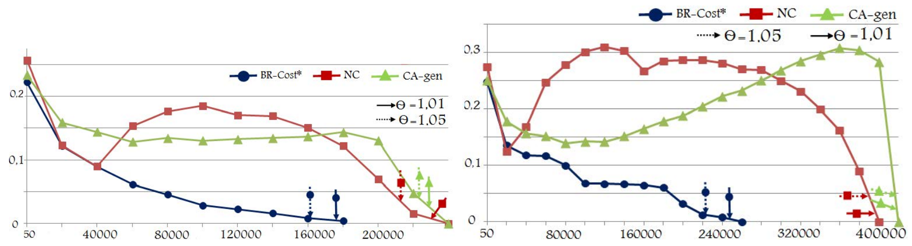
No S-sources (only R and SR). Figure 5.c shows that in all the cases BR-Cost* outperforms the other algorithms and that NC and CA-gen are less adapted to this setting, performing worse than Upper. The benefit of using BR-Cost* is bigger in the mixed distribution case (around 28% better than NC and CA-gen) 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.

Approximation Potential

We measure the potential of approximation by early stopping of the generic top-k algorithms by drawing their cost-distance curves. For space reasons, only the case of all source types is presented here.

Distance between approximate and real solution, computed with Formula (8), is measured every 2000 cost units during the algorithm's normal execution and a curve relying



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

Fig. 6. Approximation with the best k upper bound scores, all source types

these points is extrapolated. 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 (7). 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), i.e. a precision of 95%, respectively 99%. We compare the position of these points with that of the intersection between the curve and the corresponding distance.

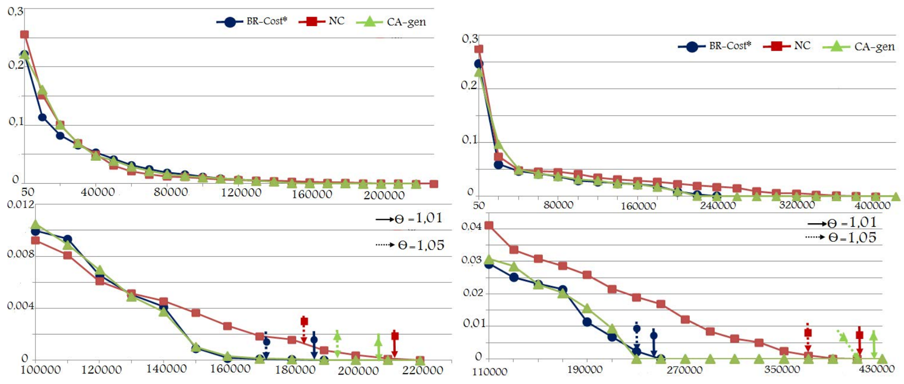
We consider two cases for the approximate solution. The first one is the set $\mathcal{U}_k$ of k candidates with the highest upper bound. This is a natural choice for the approximate solution, since $\mathcal{U}_k$ is the set of candidates that top-k algorithm focus on during execution. More precisely, 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 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 $\mathcal{U}_k$

Figure 6 presents the cost-distance curves for uniform and mixed data distributions. Final costs for algorithms may be less visible, they are already indicated in Figure 5.

For uniform distribution (Figure 6.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 6.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



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

Fig. 7. Approximation with the best k lower bound scores, all source types

at cost 160 000, while the corresponding distance of 0.05 is already reached at cost 70 000.

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 at the same precision. Comparison with specific algorithms (not presented here for space reasons) suggests that the cost of θ-approximations is 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 $\mathcal{L}_k$

For both uniform (Figure 7.a) and mixed distribution (Figure 7.b), the curves for all the algorithms are very close, stable, with good approximation potential. The sub-figure for each case presents, besides the curves, a zoom on the final part of the execution, where curves are very close. 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. 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.

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 $\mathcal{L}_k$ 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.

5 Conclusion

The BR breadth-first strategy adapts itself very well to various source type configurations and data distributions, leading to better execution cost than the other generic or specific strategies. Also, it globally has the best approximation potential among the generic strategies, with a clear advantage 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 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 that produces better total costs.

References

1. Akbarinia, R., Pacitti, E., Valduriez, P.: Best position algorithms for top-k queries. In: VLDB, pp. 495–506 (2007)
2. Badr, M., Vodislav, D.: A general top-k algorithm for web data sources. In: Hameurlain, A., Liddle, S.W., Schewe, K.-D., Zhou, X. (eds.) DEXA 2011, Part I. LNCS, vol. 6860, pp. 379–393. Springer, Heidelberg (2011)
3. Böhm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Comput. Surv. 33(3), 322–373 (2001)
4. Bruno, N., Gravano, L., Marian, A.: Evaluating top-k queries over web-accessible databases. In: ICDE, p. 369 (2002)
5. Cao, P., Wang, Z.: Efficient top-k query calculation in distributed networks. In: PODC, pp. 206–215 (2004)
6. Chang, K.C.-C., Hwang, S.W.: Minimal probing: supporting expensive predicates for top-k queries. In: SIGMOD Conference, pp. 346–357 (2002)
7. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. 66(4), 614–656 (2003)
8. Güntzer, U., Balke, W.-T., Kießling, W.: Optimizing multi-feature queries for image databases. In: VLDB, pp. 419–428 (2000)
9. Güntzer, U., Balke, W.-T., Kießling, W.: Towards efficient multi-feature queries in heterogeneous environments. In: ITCC, pp. 622–628 (2001)
10. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K.: Supporting top-k join queries in relational databases. VLDB J. 13(3), 207–221 (2004)
11. Ilyas, I.F., Beskales, G., Soliman, M.A.: A survey of top-*k* query processing techniques in relational database systems. ACM Comput. Surv. 40(4) (2008)
12. Li, C., Chang, K.C.-C., Ilyas, I.F.: Supporting ad-hoc ranking aggregates. In: SIGMOD Conference, pp. 61–72 (2006)
13. Li, C., Chang, K.C.-C., Ilyas, I.F., Song, S.: Ranksql: Query algebra and optimization for relational top-k queries. In: SIGMOD Conference, pp. 131–142 (2005)
14. Mamoulis, N., Cheng, K.H., Yiu, M.L., Cheung, D.W.: Efficient aggregation of ranked inputs. In: ICDE, p. 72 (2006)
15. Marian, A., Bruno, N., Gravano, L.: Evaluating top-*k* queries over web-accessible databases. ACM Trans. Database Syst. 29(2), 319–362 (2004)

16. Michel, S., Triantafillou, P., Weikum, G.: Klee: A framework for distributed top-k query algorithms. In: VLDB, pp. 637–648 (2005)
17. Natsev, A., Chang, Y.-C., Smith, J.R., Li, C.-S., Vitter, J.S.: Supporting incremental join queries on ranked inputs. In: VLDB, pp. 281–290 (2001)
18. Theobald, M., Weikum, G., Schenkel, R.: Top-k query evaluation with probabilistic guarantees. In: VLDB, pp. 648–659 (2004)
19. Hwang, S.W., Chang, K.C.-C.: Optimizing top-k queries for middleware access: A unified cost-based approach. ACM Trans. Database Syst. 32(1), 5 (2007)