Aspects of integration of ranked distributed data

Peter Gurský[1], Rastislav Lencses[1]

[1]*Ústav informatiky, Univerzita P. J. Šafárika*
Jesenná 5, Košice
gursky@vk.upjs.sk, lencses@science.upjs.sk

Abstract. In this paper we study integration of ranked distributed informations, which can typicaly appear in multifeature querying and retrieving top k objects in Information Retrieval, Semantic Brokering or multimedia databases. Following R. Fagin, assume that each object has m ranked attributes, one for each of m attributes, m depending on user query. For each attribute, there is a sorted list, which lists each object and its grade under that attribute, sorted by grade (highest grade first). Each object is assigned an overall rank that is obtained by combining the attribute grades using a learnable monotone aggregation function depending on user preferences. We propose new algorithms for finding top k objects and compare them to existing ones in experiments on real and also artificial data. This experiments have shown that top k objects appear much faster than confirmation that these are exactly the top k objects. Based on this, we proposed and tested heuristic algorithms. We tested succesfully algorithms for finding approximately top k answers with additive precision factor.

Keywords: integration distributed information, middleware, aggregation

1 Introduction

Today is becoming important for Semantic web, browsers, relational and object database systems to be able to access multifeature data, such as video, images, audio, text and another objects with many different attributes. Such attributes are typically fuzzy. We introduce following example.

In the area of information retrieval user query consists of terms. Query terms are attributes of documents. The search system retrieves documents relevant to user query. The grade of query term for document typically depends on number of this term in document and on importance of this term in collection of documents. Document must be ranked with some aggregation function according to query terms.

We try to retrieve top k objects with highest overall grades in the introduced example. Every object has attributes which are probabilistically independent and overall grade of the object is computed as an aggregation function of these attributes. Typical input data for this problem are m lists of objects sorted by attribute.

Fagin [4, 6] introduced “Fagin’s algorithm” which solve this problem first time. Fagin et al. [5] presented “threshold” algorithm. Nepal et al. [9] defined algorithm that is equivalent to threshold algorithm. Güntzer et al. [2] define “quick-combine”

algorithm with heuristic rule that determines which sorted list should be preferred. Natsev et al. [10] generalized this problem as an arbitrary joins in databases.

In chapter 2 we describe formal model of data. In chapter 3 we present known algorithms, in chapter 4 are described our algorithms. Chapter 5 consists of results of experiments. Chapter 6 concludes our paper.

2 Model

Assume we have finite set of objects. Cardinality of this set is N. Every object has m attributes x_1, .. x_m. All objects are in lists L_1, .. L_m, each of length N. Objects in list L_i are ordered descending by value of object in atribute x_i. We can define two functions to access objects in lists. Let x be an object. Then $s_i(x)$ is grade (or score, rank) of object x in list L_i and $r_i(j)$ is object in list L_i in j-th position. Lists can be accessed in two ways. First one is sorted (sequential) access. Grade of objects are obtained by proceeding through the list sequential from the top. The second mode of access is random access. Here, we request the grade of object x in list L_i and obtain it in one random access.

We have also monotone aggregation function F, which combine grades of object x from lists L_1, .. L_m. Overall value of object x we denote as S(x) and it is computed as $F(s_1(x), .. s_m(x))$.

Our task is to find top k objects with highest overall grades. We also want to minimize time and space. That means we want to use as low sorted and random accesses as possible.

3 Algorithms

There are many algorithms in this area. First we will interest in so-called naive algorithm.

This algorithm looks at every item in each of the m sorted lists, computes the overall rank of these objects, sorted them by rank and returns the top k answers. As this algorithm computes rank of all objects, it can be inefficient. There are algorithms, which can obtain top k answers without looking for every object.

3.1 The Generalized Version of the Threshold Algorithm

Let $z = (z_1,\ldots z_m)$ be a vector, which assigns each i=1, … m the position in list L_i last seen under sorted access. Let H be a heuristics that decides which list (or lists) should be accessed next under sorted access (notice that heuristics can change during the computation). Moreover, assume that H is such, that for all $j \leq m$ we have $H(z)_j \geq z_j$ and there is at least one $i \leq m$ such that $H(z)_i > z_i$. The set $\{i \leq m: H(z)_i > z_i\}$ we call the set of candidate lists for next sorted access. Note that if $H(z)_i > z_i+1$, then at least two sorted access can be done in list L_i, of course preserving the order of elements of L_i.

Different algorithms differ in H (or a family of heuristics H) and also in treating candidates for next step – access parallel [p] all candidates (case 2a) or randomly [r] choose only one candidate (case 2b).

Now we describe a class of algorithms TA(H, [p|r]) – clones of TA from [5].

z:=0,

1. update H,
2. Case 2a. Do sorted access in parallel to each of the sorted lists to all positions between z_i and $H(z)_i$ respecting order of lists. Put $z_i = H(z)_i$.

   Case 2b. Randomly choose one i such that $H(z)_i > z_i$, Do sorted access to list L_i and put $z_i := z_i + 1$ and for all other j, z_j does not change.
3. As an object x is seen under sorted access in some list, do random access to the other lists to find the grade $s_i(x)$ of object in every list. Then compute the grade $S(x) = F(s_1(x), .. s_m(x))$ of object x. If this grade is one of the k highest we have seen, then remember object x and its grade S(x) (ties are broken arbitrarily, so that only objects and their grades need to be remembered at any time).
4. For each list L_i , let $u_i = s_i(r_i(z_i))$ be the grade of the last object seen under sorted access. Define the *threshold value* τ to be $F(u_1, …., u_m)$. As soon as at least k objects have been seen whose grade is at least equal to τ , then halt, otherwise go to step 1.
5. Let Y be a set containing the k objects that have been seen with the highest grades. The output is then the graded set $\{(x, S(x)) : x \in Y\}$.

Theorem 1. If the aggregation function F is monotone, then TA(H,[p|r]) correctly finds the top k objects.

Proof. Literally the same as in [4], the only difference is that the threshold is computed along the line zi.

The original Threshold algorithm (see [4] and [5]) works with heuristics $H(z)_i = z_i + 1$ and variant [p] that is parallel accessing sorted next entry in each list.

3.2 Properties of Aggregation Function F and Distribution of Data - The Quick-Combine Algorithm ($\partial F/\partial x * \Delta x$)

All clones of the TA algorithm that we will describe use heuristics to determine which list should be accessed next under sorted access. First idea came from [2] where authors wrote:

Obviously, there are two ways to make

$$S(x) = F(s_1(x), .. s_m(x)) \geq \tau$$

hold: to increase the left side or to decrease the right side. Rapidly decreasing of τ causes that algorithm quickly confirms the correctness of the top k list.

So basic parameters we will take into account in defining our strategies are:

Partial derivative of F from the left

$$\left(\frac{\partial F}{\partial x_i}(x_1,...,x_m) \right)^-$$

and the value of data in list Li currently read zi or the expected change of data in p steps ($s_i(r_i(z_i - p)) - s_i(r_i(z_i))$).

Güntzer et al. in [2] introduced the Quick combine algorithm with following heuristics: After p parallel accesses in each list, do sorted access in such lists whose values descend rapidly. This means our threshold descends rapidly too and we can get confirmation of top k objects faster. This holds in some situations, which depends on aggregation function and histogram of data. As a criterion which list is suitable, authors choose next equation:

$$\Delta_i = \left(\frac{\partial F}{\partial x_i}(x_1,\ldots,x_m) \right)^{-} * (s_i(r_i(z_i - p)) - s_i(r_i(z_i))$$

The constant p is some suitable natural number. Every step in this algorithm we compute Δ_i for every list and we choose randomly list with the biggest value of Δ_i (the variant [r] of the heuristics). We also tested the variant of this heuristisc, in which we do parallel access to all of the candidate lists (not randomly). This alternative looks to be faster in most of tests.

This algorithm is also correct for monotone aggregation function F. We symbolically denote this algorithm as (∂F/∂x*Δx) heuristics algorithm.

4 Our Algorithms

Our approach is in some sense similar and in some sense opposite to previous one. We discuss several possibilities. We can get the top k objects faster such way, at least in some situations which we will discuss later. We have two criteria to choose which list is suitable.

4.1 Slowest F Descent – Parallel ∂F proportional

First possibility, which was studied in more detail, is to follow in parallel access proportional to all partial derivatives from the left

$$\partial_i = \left(\frac{\partial F}{\partial x_i}(s_1(r_i(z_i)),\ldots,s_m(r_i(z_i))) \right)^{-} \quad (1)$$

The criterion (1) computes partial derivation of *F* and value of this derivation in the point ($s_1(r_i(z_i))$, ..., $s_m(r_i(z_i))$). The heuristics for calculation of candidate lists says

$$z_i := z_i + \partial_i$$

of course if $z_i + \partial_i$ is not bigger than z_i+1, then do not access L_i but accumulate ∂_i until it exceeds 1 and then the list is accessed. In the case *F* is linear, derivation is constant c_i. We symbolical denote this algorithm as (∂F/∂x)-heuristics algorithm and provide some experimental comparisons.

4.2 Randomly go for greatest ∂F/∂x*x

This is a variation of quick combine algorithm in [2], which randomly chooses list with the greatest (∂F/∂x*Δx). Instead of following Δx factor we choose x-factor that is the biggest value in list multiplied with the least descent.

$$c_i = \left(\frac{\partial F}{\partial x_i}(s_1(r_i(z_i)),\ldots,s_m(r_i(z_i))) \right)^{-} * (s_i(r_i(z_i)) \quad (2)$$

The criterion (2) computes left partial derivation of F and value of this derivation in the point $s(r(z))$ and multiplicates it with value in the point $s_i(r_i(z_i))$ in L_i and randomly chooses one of lists with biggest value. In our experiments we refer to this algorithm as to the (∂F/∂x)*x-heuristics algorithm.

4.3 x/Δx - Switching Heuristics

The original motivation of [2] was to decrease faster the right hand side of

$$S(x) = F(s_1(x), .. s_m(x)) \geq \tau$$

And go in the direction of greatest decrease of (∂F/∂x*Δx). Now the idea is to simultaneously try to keep left hand side big – as the (∂F/∂x)*x-heuristics algorithm is doing and simultaneously decrease the right hand side as the (∂F/∂x*Δx) -heuristics algorithm is doing. Note that both use random choice of candidate list. This algorithm is an example where the heuristics change during the computation, e.g. for switching rate 1:1, in even steps follow (∂F/∂x)*x-heuristics and in odd steps follow (∂F/∂x*Δx)–heuristics.

4.4 Two-Phased Algorithm

We tried to find suitable heuristics to estimation time and probability of existence final top k list without confirmation.

(∂F/∂x)*x algorithm is able to find top k objects without confirmation faster than other algorithms. (∂F/∂x*Δx) is the fastest algorithm in decreasing value of τ. Fast decreasing of τ causes that algorithm quickly confirms the correctness of the top k list. We tried to use advantage of both algorithms to construction of the two-phased algorithm. In the first phase of this algorithm we use (∂F/∂x)*x heuristics. After b steps we switch to (∂F/∂x*Δx) algorithm (we switch only ones). The correctness of the top k list is preserved as both algorithms preserve the correctnes of the top k list. The number b can depend on k, m, F and distribution of data. We observed that x/Δx -switching heuristics performs better than two-phased algorithm for tested values of b.

4.5 Additive Approximation of Top k Objects

For many applications and many data types we have only finitely many classes of object classifications and also users often do no need real valued classification of object – instead some 7±2 classes suffice. Moreover this idea has shown in our experiments to improve computational efficiency. For achieving this we have to use the [5] of approximative top k objects, nevertheless we have to consider not multiplicative but additive approximation factor.

Let ε > 0 is given. Define an ε-*additive-approximation to the top k answers (for F over database* ***D*** *)* to be a collection of k objects (and their grades) such that for each y among these k objects and each x not among these k objects,

$$S(y) + \varepsilon \geq S(x).$$

We can modify all clones of TA algorithm to find an ε-additive approximation to the top k answers by modifying the stopping rule in the step 4 of TA(***H***,[p|r]) to say "As soon as at least k objects have been seen whose grade is at least equal to τ - ε , then halt." (refrasing [5]). Let us call this approximation algorithm TA(***H***,[p|r])$^{\varepsilon}$. A similar result to that of [5] can be proven

Theorem. *Assume that* ε > 0 *and that the aggregation function F is monotone. Then both* TA(***H***,[p|r])$^{\varepsilon}$ computed with F and TA(***H***,[p|r]) computed with ε *rounding of F correctly finds an* ε*-additive-approximation to the top k answers for F.*

Proof: Similar to that of [5].

4.6 An Improvement in the Case of User Constraints

User could specify constraints for values of atributes in lists L_1, .. L_m indepedently of aggregation function *F*. These constraints are possible to use in some improvements applicable in all algorithms.

We define constraint as an interval for values of every attribute in his domain. Valid values of attributes are within this interval only. When user specifies lower bound of values of list L_i, an algorithm will simply does not do sorted access to this list below specified bound. Specification of upper bound can also speed up an algorithm. We need to find a supremum of valid values. If the system does not allow access to value of attribute in specified order arbitrary, an algorithm will extend its heuristics by inserting this list to candidate set whenever until it reaches supremum (with sorted access). Otherwise we can use binary search to find this supremum. In the both cases we do not use random accesses for objects with values out of constraints. The objects with invalid values we do not put to the top k list, also in the case we read its values with random access.

5 Experiments

We experimented with real data that come from randomly generated queries in information retrieval system with support of relational database, which was designed similar to [7].

We used different ways for weighting of occurrences terms in documents. Here is typical formula for compute weight of occurrence term t_j in document d_i :

$$w_{ij} = l_{ij} \cdot g_j \tag{3}$$

In equation (3) l_{ij} is local weight of term t_j in document d_i and g_j is global weight of term t_j in whole collection of documents. Local weight can be computed as number of occurrences of term in document. We can denote this number of occurrences as tf_{ij}. Another possibility is to decrease this factor and use $log(1+ tf_{ij})$. In the case there are controlled vocabularies is possible to use binary weights: the weight l_{ij} is 1 if the term t_j occurs in document d_i and l_{ij} is 0 otherwise. Global weight g_j is used to compute

importance of term t_j in whole collection. Typically this weight should by low for connectives such "and", "or" and high for good discriminators such nouns with low occurrence in whole collection. Some known examples include inverse document frequency, normal global weight and third global weight (see [1]).

We used different combinations of local and global weights in order to generate 6 sets of benchmark data.

To measure all particular experimental results we compare the average values from 6 sets of these benchmark data with the use of aggregation functions with randomly chosen weights for each list. Histograms of data are exponential – there is very low number of objects with high values and a lot of objects with low values.

5.1 Results

As we can see in Figure 1, the total number of all accesses is the best for x/Δx switch algorithm. It is very interesting that this algorithm is better than both algorithms of which is created. We can see that the strategy of decreasing the threshold value or the increasing the values of the top-k list (both separately) has lower power than to use of both strategies.

In Figure 2 we can see number of accesses (both random and sorted accesses) needed to find top *k* objects without confirmation of correctness. Figure 3 shows that the price of correctness confirmation of the top-k objects list can be as high as 24 times greater.

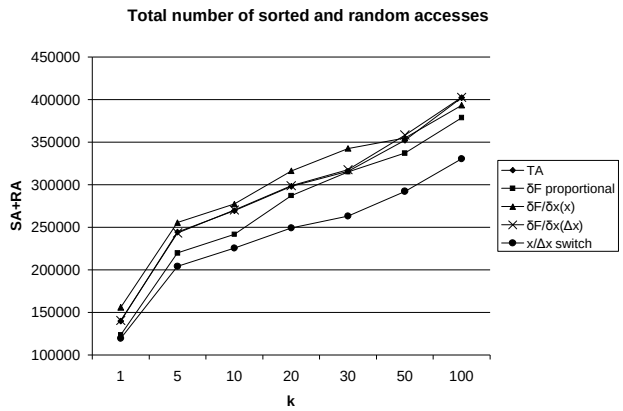

Figure 1 Total number of random and sorted access, less is better

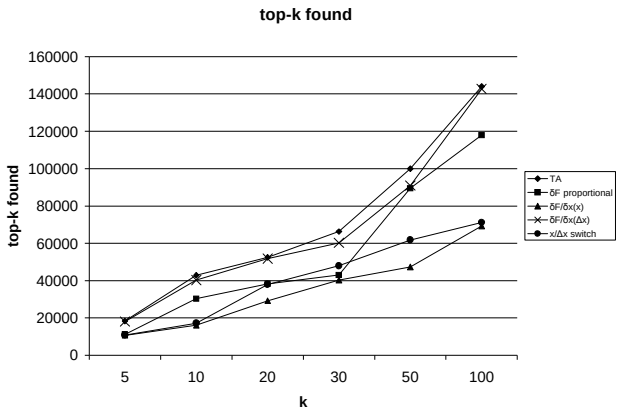

Figure 2 Number of all accesses to find top k objects without confirmation, less is better

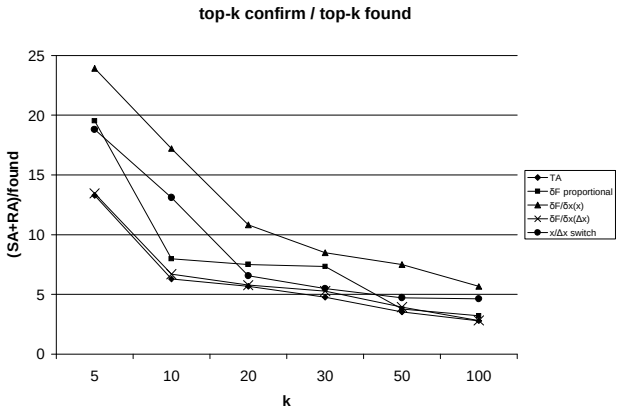

Figure 3 The ratio between found top k objects and confirmation of top k objects

In Figure 4 we can see the average improvement factor of all accesses when we use the 0,1-additive approximation with the use of the same aggregation function for each experiment. For example, $\partial F/\partial x * \Delta x$ needs 47 times lesser of all accesses than without applying approximation. All other algorithms need about 25 times lesser of all accesses.

In Figure 5 we experimented with variable *m*. We can see that behavior of all algorithms is influenced with different aggregation functions applied for different test (because of *m* is changed, *F* must be changed too). Our function F is weighted mean with sum of random coeficients equal to 1. This causes that weights of the function *F* are decreasing with growing *m*. The x/Δx switch algorithm seems to be better esspecially for growing *m*.

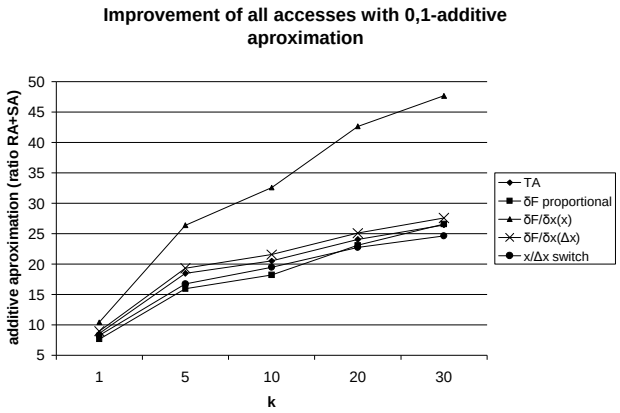

Figure 4 Improvement of number of all accesses (sorted and random) with additive approximation, more is better

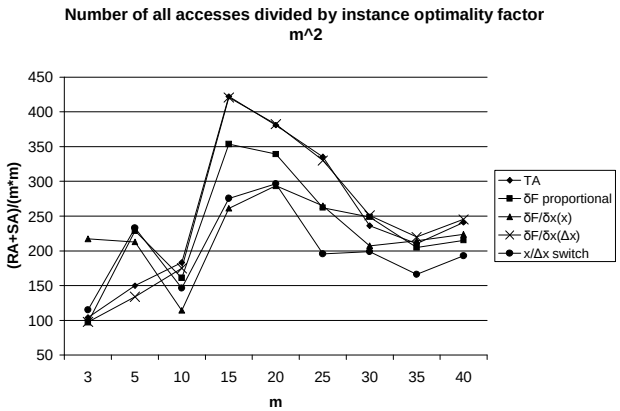

Figure 5 A comparison of accesses modulo instance optimality factor m^2 for k=10

6 Conclusion

In this paper we have studied user dependent integration of ranked distributed informations, which can typically appear in multifeature querying and retrieving top k objects in Information Retrieval, Semantic Brokering or multimedia databases.

We started from model of R. Fagin and proposed several new algorithms for correctly finding top k objects, where several new heuristics as to which list should be accessed next under sorted access were applied. We have shown that such heuristics lead to some speedup of TA of Fagin, Loten and Naor, and also of ($\partial F/\partial x * \Delta x$)

heuristic of Güntzer, Balke and Kiessling, especially on data distributions usual in IR. We compared them to existing ones in experiments on our own benchmark data.

Moreover, this experiments have shown that top k objects appear much faster than confirmation that these are exactly the top k objects. Based on this, we have proposed the two-phased algorithm. We observed that our switching algorithm performs better than two-phased algorithm (and better than any one).

We tested succesfully with algorithms for additive approximation of top k which also speeds up the run time and gives user a granulated classification of answers.

Moreover we use user dependent aggregation function which can be learned and also consider the problem with variable number of lists and/or features of our query. Our experiments show that our algorithms are also better when performance measures are divided by the factor m^2 from Fagin's estimation of instance optimality with constant number of lists m.

References

1. Berry, M. W., Browne, M.: Understanding search engines, SIAM, Philadelphia, 1999
2. Güntzer, U., Balke, W., Kiessling, W.: Optimizing Multi-Feature Queries for Image Databases, Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000
3. Fagin, R.: Combining Fuzzy Information: an Overview, ACM SIGMOID Record 31, Database principles column, June 2002, pages 109-118
4. Fagin, R.: Combining fuzzy information from multiple systems. J. Comput. System Sci., 58:83-99, 1999
5. Fagin, R., Lotem, A., Naor, M.: Optimal Aggregation Algorithms for Middleware. In Proc. 20th ACM Symposium on Principles of Database Systems, 2001, pages 102-113
6. Fagin, R.: Combining fuzzy information from multiple systems. In 15th ACM Symposium on Principles of Databases Systems, pages 216-226, Montreal, 1996
7. Grossman, D., Frieder, O.: Information Retrieval: Algorithms and Heuristics, Kluwer Academic Publisher, Massachusetts, 2000
8. Horváth, T., Vojtáš, P.,: GAP-rule discovery for graded classification, preprint 2004, 12 pages
9. Nepal, S., Ramakrishna, M. V.: Query processing issues in image (multimedia) databases. In Proc. 15th International Conference on Data Engineering, pages 22-29, 1999
10. Natsev, A., Chang, Y-C., Smith, J.R., Li, C-S., Vitter, J.S.: Supporting incremental join queries on ranked inputs. In Proc. 27th Very Large Databases Conference, pages 281-290, Rome, Italy, 2001
11. Pokorný, J., Vojtáš, P.: A data model for flexible querying. In Proc. ADBIS'01, A. Caplinskas and J. Eder eds. Lecture Notes in Computer Science 2151, Springer Verlag, Berlin 2001, 280-293
12. Gurský, P., Lencses, R., Vojtáš, P.: Algorithms for User Dependent Integration of Ranked Distributed Information, preprint to ISWC 2004