ALGORITHMS FOR USER DEPENDENT INTEGRATION OF RANKED DISTRIBUTED INFORMATION

Peter Gurský[1], Rastislav Lencses[1], Peter Vojtáš[2]

Introduction

In this paper we study integration of ranked distributed information in multifeature querying and retrieving top k objects. We propose several variants of the threshold algorithm and compare them to existing ones. These 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.

E-Government is generally defined as the use of information and communication technology in public administrations in order to improve public services and democratic processes. In order to use this possibility more effectively, it 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 and formats. Such attributes are typically fuzzy (see e.g. [3]). We introduce following motivating example.

Example

In the area of *electronic democracy* a user (citizen, a subject of law) is assumed to have greater access to data and information on the net.

A user might look for a locality to build a facility. His requirements are low price, close distance and flooding safety. All of these are distributed over the net and user's requirements are inherently fuzzy – ranked by a real number from [0, 1]. For a decision, user might calculate overall fitness by a weighted sum, e.g.

$$\frac{10 * flood + 5 * cheap + close}{16}$$

We would like to retrieve top k objects with highest overall grades. Every object has attributes which are probabilistically independent. Typical input data for this problem are ***m*** lists of objects sorted by attribute.

This problem was formulated and first studied by R. Fagin and in [4, 6] introduced "Fagin's algorithm" which gives a first solution. Fagin et al. [5] presented "threshold" algorithm, which is provably instance optimal. S. Nepal et al. [9] defined algorithm that is equivalent to threshold algorithm. U. Güntzer et al. [2] define "quick-combine" algorithm with heuristic rule

[1] Institute of Computer science, University of P. J. Šafárik, Košice, {gursky, lencses}@upjs.sk
[2] Dpt. of Software Engineering, Charles University, Prague, vojtas@ksi.ms.mff.cuni.cz

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

Model

Following R. Fagin [3], assume we have finite set of objects. Cardinality of this set is N. Every object x has m attributes $x_1, ..., x_m$. The attribute is a quantifiable property of an object; the values of attributes are real numbers. All objects are in lists $L_1, ... L_m$, each of length N. Objects in list L_i are totally ordered descending by value of object in attribute 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 (direct) access. Here, we request the grade of object x in list L_i and obtain it in one random access.

We have also a monotone aggregation function F, which combine grades of object x from lists $L_1, ..., L_m$. We require that function F is monotone in all arguments and continual from the left. This function is in the input in all mentioned algorithms. 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.

Algorithms

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

This algorithm looks at every item in each of the m sorted lists, computes the overall rank of these objects, sorting 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.

The Generalized Version of the Threshold Algorithm

Let $z = (z_1,...z_m)$ be a vector, which assigns each $i = 1, \ldots m$ the position z_i 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 candidate set 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. We denote the heuristics with parallel access to all candidates from the candidate set (case 2a) with [p]. Similarly we denote the heuristics with access to exactly one random candidate (case 2b) with [r].

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

0. 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 ones we have seen, then remember object x and its grade $S(x)$ (ties are broken arbitrarily, so that only k-many 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 i^{th} grade of the last object seen under sorted access in L_i. Define the *threshold value* τ to be $F(u_1, \ldots, u_m)$. 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 $\{(x, S(x)) : x \in Y\}$.

Definition 1. Let K be a set containing top k objects. Algorithm finds the top k objects correctly when

$$\forall x \in K \, \forall x' \notin K \quad S(x) \geq S(x')$$

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 [5], the only difference is that the threshold is computed along the line z_i. Main idea is that algorithm finds top k objects correctly, if the value of the threshold is less or equal to the object in the top k list with the smallest value what is guaranteed by step 4 of algorithm TA(H, [p|r]).

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. Hence we denote this algorithm as TA($z_i + 1$, [p]).

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. We start with the algorithm from the work of Güntzer, U., Balke, W., Kiessling, W. [2].

Every object x from top-k list must satisfy correctness, which can be expressed as in previous part

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

Güntzer et al. made following observation: There are two ways to satisfy this expression during the computation: 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.

In [2] authors choose two basic parameters into account. The first parameter is the partial derivative of F from the left in the point $s_1(r_1(z_1)),...,s_m(r_m(z_m))$:

$$\partial_i = \left(\frac{\partial F}{\partial x_i}(s_1(r_1(z_1)),\ldots,s_m(r_m(z_m))) \right)^- \quad (1)$$

The second parameter is the expected change of data in p steps, where the constant p is some suitable natural number

$$(s_i(r_i(z_i - p)) - s_i(r_i(z_i))$$

The only necessary condition is F should have all partial derivatives from the left.

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 the fastest. This means the 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 H which list is suitable, authors choose i maximizing next equation:

$(\partial F/\partial x)*\Delta x$

$$\Delta_i = \partial_i \cdot (s_i(r_i(z_i - p)) - s_i(r_i(z_i))) \quad (2)$$

In [2] compute Δ_i in every step for every list L_i. Then put in the candidate set the lists with the maximum value of Δ_i. The original algorithm [2] uses the variant [r] of the heuristic.

We have also tested the variant [p] of this heuristics. This alternative was faster in the most of our tests.

This algorithm is also correct for monotone aggregation function F. We symbolically denote this algorithm as $((\partial F/\partial x)\cdot \Delta x)$ heuristics algorithm.

Our Algorithms and heuristics

Our approach is in some sense similar and in some sense opposite to previous one of [2]. 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.

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. The criterion (1) computes partial derivation of F and value of this derivation in the point $(s_1(r_1(z_1)), ..., s_m(r_m(z_m)))$. The heuristics for calculation of candidate set 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.

Randomly go for greatest *(∂F/∂x)*x*

In [2], algorithm randomly chooses list with the greatest *((∂F/∂x)· Δx)*. Now, instead of Δx we choose x. This is a variation of quick combine algorithm. We choose the list which contributed to aggregation function F with last read value at most. This heuristic searches the objects with the biggest overall value (increases left hand side of correctness equality, opposite to [2] which decreases right hand side).

*(∂F/∂x)*x*

$$c_i = \partial_i * (s_i(r_i(z_i)) \quad (3)$$

The criterion (2) computes left partial derivation of F and value of this derivation in the point $s(r(z))$ and multiplies 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.

***x*/Δ*x* - 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)**Δ*x*. In 4.2 we followed the idea to increase left hand side. Now the idea is to do simultaneously both, i.e. try to keep left hand side big – as the *(∂F/∂x)*x*-heuristics algorithm is doing and simultaneously decreases the right hand side as the *(∂F/∂x)**Δ*x*-heuristics algorithm of [2] is doing. Note that both use random choice of candidate set. 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)**Δ*x*–heuristics.

Additive Approximation of Top *k* Objects

There is yet another chance to accelerate algorithms. Let $\varepsilon > 0$ is given. Define an *ε-additive-approximation to the top k answers* 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,

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

The idea of approximation comes from [5] in which authors used the multiplicative approximation. Our solution is suitable when it is necessary or suitable to define an absolute error ε and not a ratio (e.g. in cases when object with overall value S(x) smaller than ε are no more relevant).

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 $\tau - \varepsilon$, then halt" (rephrasing according

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

Proof: Similar to that of [5].

Idea of the additive approximation improves computational efficiency.

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].

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. Such distribution is typical in the information retrieval area.

Results

In Figure 1 we can see number of accesses (both random and sorted accesses) needed to find top k objects without confirmation of correctness. When algorithm has the final top k list, it does not know it. The algorithm has to confirm this final top k list. It means, that the threshold must to go down until its value is less or equal to value of all objects in the top k list. Figure 2 shows that the price of correctness confirmation of the top-k objects list can be as high as 24 times greater. The total time complexity is propositional to value form Figure1 multiplied the ratio from Figure 2 (here $x/\Delta x$ - Switching Heuristics is the best with improvement about 20%).

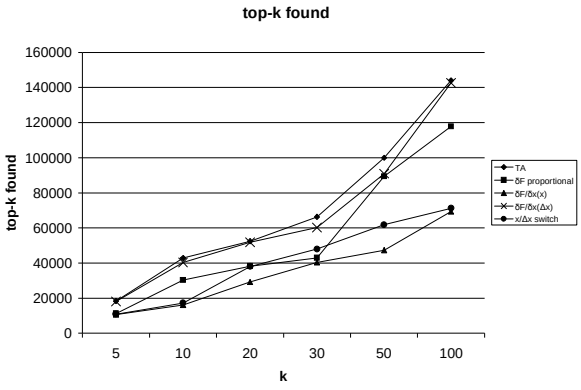

Figure 1. Number of all accesses to find top k objects without confirmation, less is better

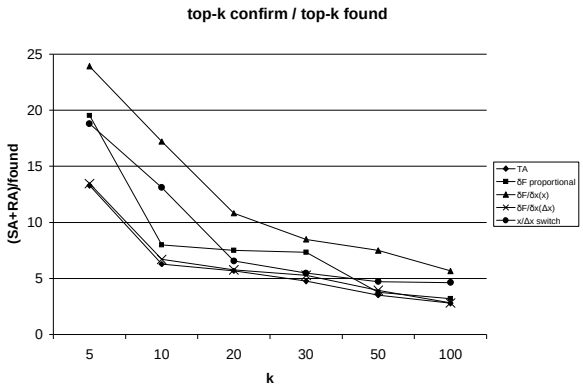

Figure 2. The ratio between found top k objects and confirmation of top k objects

Further we have measured 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.

Querying with fixed number of attributes – features (fixed m) is typical for multimedia databases. For such models in [5, Theorem 6.4] an instance optimality of TA algorithm is proved, up to a constant factor m^2. For information retrieval queries with variable number of terms (terms are playing the role of features) are typical. In this case we do not know whether TA is or not instance optimal.

We experimented with variable m. Our function F is a weighted mean with sum of random coefficients equal to 1. This causes that weights of the function F are decreasing with growing m. The $x/\Delta x$ switch algorithm seems to be better especially for growing m.

Conclusion

In this paper we have studied user dependent integration of ranked distributed information, 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 Güntzer et al. 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. In initial experiments we have shown that such heuristics lead to some speedup of TA of Fagin, Lotem 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. In future, a detailed comparative analysis in order to provide more accurate results validating our algorithms is planned.

Moreover, this experiments have shown that top k objects appear much faster than confirmation that these are exactly the top k objects. This opens a possibility for a two-phased algorithm in future. In preliminary tests, we observed that our switching algorithm performs better than two-phased algorithm (probably our heuristics have to develop further).

We tested successfully 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.

Announcement. This research was partially supported by the Czech Program "Information Society" under project 1ET10030517 and Slovak project VEGA 1/0385/03.

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