Towards Better Semantics in the Multifeature Querying**

Peter Gurský

Institute of Computer Science, Faculty of Science
P.J.Šafárik University Košice
Jesenná 9, 040 01, Košice, Slovakia
gursky@upjs.sk

Abstract. Nowadays the natural requirement of users is to retrieve the best answers for the criteria they have. To explain, what kind of objects user prefers, we need to know, which values of properties are suitable for the user. We assume that each property is possibly provided by an external source. Current algorithms can effectively solve this requirement, when the sources have the same ordering as the user preferences. Commonly, two users prefer different values of a given property. In this paper we describe how we can consider this feature.

Key words: multifeature querying, top-k objects, aggregation, fuzzy function

1 Introduction

Many times users want to find the best object or top k objects in the possible huge set of objects. The decision which object is better than the other, is based on the properties of the objects. Typical objects are job offer, document, website, picture, book, presentation, conference, hotel, vacation destination etc.

Such objects have typically some properties (attributes). Users can search and decide, which objects are the best using these properties. Such searching is made by a multifeature deciding.

The property of an object is, typically, one of four types. First type of properties is boolean or yes/no property. Examples can be: work at home, if somebody is married, breakfast included, aspect at the sea, Springer proceedings etc. Second type of properties is properties, that are graded in some way to finite number of classes. Typical properties are: number of stars of hotels, quality of an article, level of education etc. Third type is real or integer number, for example: salary, price, number of pages, properties in multimedia databases, date etc. The last type of properties is text. In multifeature querying we can use this kind of attribute to reduce searching space (user could search only IT jobs), especially,

** This work was partially supported by the project 'Štátna úloha výskumu a vývoja "Nástroje pre získavanie, organizovanie a udržovanie znalostí v prostredí heterogénnych informačných zdrojov" prierezového štátneho programu "Budovanie informačnej spoločnosti".'

V. Snášel, K. Richta, J. Pokorný (Eds.): Dateso 2006, pp. 63–73, ISBN 80-248-1025-5.

when such a property is organized in some hierarchical structure e.g. ontology. We can also derive some other properties from a text. These properties should be one of the three previous types. For example from the human first name we can learn a gender, from the name of a town we can find out a distance from a specific location.

Useful condition is to assume that each property is provided by a possibly remote source. For example to find out distances, we can use servers of traffic companies. Information about free places in hotels can be provided by a server, which collects such information. We can say that sources are distributed, also when the information are on the same computer, but stored in different repositories (RDBS, Ontology, files). This condition is suitable also in cases, when we want to combine several search methods and aggregate them to one list of the best objects. In this case the sources are typical input streams. In the rest of the paper we assume that the properties of objects are provided by distributed sources.

The problem is, how to specify, whose objects are the best. First approach to this problem is to use a monotone aggregation function. Ronald Fagin in 1996 introduced "Fagin's algorithm", which solves this problem first time in [6]. Fagin et al. [7] presented "threshold" algorithm that made the search much faster. Güntzer et al. [1] defined "quick-combine" algorithm using first heuristic. Other heuristics was presented by P. Gurský and R. Lencses [10]. M.Vomlelová and P. Vojtáš [9] propose a probabilistical heuristic. All these solutions use two types of accesses - sorted access and random access. The sorted access is a sequential access to a sorted list. Using this type of access the grades of objects are obtained by proceeding through the list sequentially from the top. Random access returns the property value for a specified object. Further papers deal with the situation, when some kind of accesses is slow or impossible. There was defined a "combined algorithm" in [7] that count with the prices of accesses. In the same paper authors propose algorithm NRA (no random access) that does not use the random access. Güntzer et al. [2] present algorithm "Stream-combine" that uses some heuristics. Combination of last two approaches is "3P-NRA" algorithm presented by P. Gurský [13] with a new heuristic holes. All three approaches use only sorted access.

The second way to specify the objects to retrieve is a skyline. In the skyline, there are objects that are pareto optimal. An object is pareto optimal, if it is not dominated by any other object. Object x dominates object y if x has greater or equal score in all properties and is strictly greater in at least one property than y. Skyline was firstly presented by S. Börzsönyi et al. [8]. Authors put a proposed algorithm to the database query processor. First solution in the field of multifeature querying was presented by W. Balke et al.[3]. In [4] authors combine features of skylining and aggregation functions as "multi-objective retrieval". In this approach we can specify several monotone aggregation functions. The final set of objects is a skyline with respect to the values of the aggegation functions. In this case, values of aggegation functions are construed as the properties for a skyline. W. Balke at al. [5] propose an algorithm for the case of weak pareto

optimality. It differs from "normal" pareto optimality by partial ordering on domains of the properties. In this case object x dominates object y, if x is strictly greater in at least one property and there is no property such that y has greater score than x.

P.Gurský and T.Horváth in [12] use induction of generalized annotated programs (IGAP) to learn monotone graded classification described by fuzzy rules. Fuzzy rules play role of the monotone aggregation function. As input for this approach is classification of several objects by a scale from the worst to the best.

All current solutions assume, that the sources send their property information ordered from the best value to the worst value. None of these solutions allow the user to specify, which values of a property are better than the other. In this paper we discuss about the possibility of preference specification and effective retrieval of top k objects.

Imagine that we want to find a hotel in a city and we can decide using the properties "distance from the centre", "price" and "number of stars". The main problem is that we cannot universally set the orderings from the best distance to the worst distance or from the best number of stars to the worst. One user can say that the best is to sleep in the centre in the low quality and cheap hotel, a second one may prefer hotels in the country (far from the centre) and with highest possible number of stars. The other one prefer rather quiet suburb place and accepts (and needs) traveling to the centre, but not very long. So he or she prefers middle values of the distance. It is possible that 90 % of people prefer cheap hotels before the expensive ones, but, for example, if somebody is on the business travel and the accommodation is paid by the company, he or she may prefer luxury expensive hotels.

Such properties with unknown default orderings are quite common, also in the case of other types of objects. To explain the preferences by a user we can use fuzzy functions. The explanation can be done by a different method too. Since the fuzzy function is an explicit assignment, we assume that there is a transformation to fuzzy functions. We will consider four types of fuzzy sets (see Figure 1). On the axis x, there are property values e.g. price or number of stars. On the axis y we have preferences of a user, where 1 means strong preference and 0 means no preference. Imagine that we want to explain our preference to the property "distance from the center". If we prefer hotels in the center, our fuzzy function will look like the fuzzy function A on the figure 1. Fuzzy function B means that we prefer hotels far away from the center, fuzzy function C means our preference for hotels in suburb. Fuzzy function D means that we want to be right in the center or out of the city. Fuzzy function A in the case of the price property means, that we prefer cheap hotels, and in the case of number of stars it means our inclination to low quality hotels (low number of stars).

These 4 types of fuzzy functions were strong enough to use for user specification in all domains we considered. It is quite unusual to say, for example, that we prefer middle values but not exactly in the middle, thus the fuzzy function should have two local maximums and three local minimums. The question, if

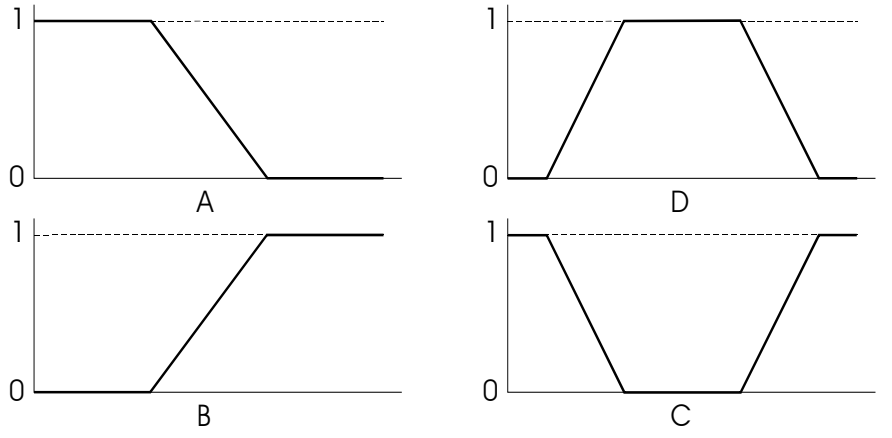

Fig. 1. Fuzzy functions

these 4 types of fuzzy function are enough, is more philosophical. We assume that it is enough.

Interesting way to define user fuzzy functions is to learn them from the evaluation of objects in sample collection. The evaluation can be done in some scale from the best to the worst. It is possible to learn them by the system QUIN (QUalitative INduction). The approach was suggested by Šuc [15].

When the fuzzy functions are defined, the same evaluation of objects can be used by system IGAP to learn monotone graded classification. If we will be able to find top k objects using these fuzzy functions the system presented in [12] should lead to better results. The technique presented in [12] uses only linear regression to learn fuzzy functions so it works good if a user has preferences of properties described by fuzzy functions of type A and B from Figure 1.

In this paper we try to propose a way to extend the functionality of multifeature querying. In the next sections we describe two main approaches to the distributed multifeature querying. Then we propose new extensions of these algorithms.

2 Model

First of all we need to specify basic model and determine useful terms. Assume we have a finite set of objects. Cardinality of this set is n. Every object x has m attributes $x_1, \ldots, x_m$. All objects (or identifiers of objects) are in lists $L_1, \ldots, L_m$, each of length N. Objects in list L_i are ordered descending by values of attribute x_i. As we said, we can define two functions to access objects in lists. Let x be an object. Then $r_i(x) = x_i$ is a grade (or score, rank) of object x in the

list L_i and $s_i(j)$ is an object in the list L_i at the j-th position. Using the function $r_i(x)$ we can realize the *random access* to the lists, i.e. for object x we retrieve the value of i-th property. The second type of access we will use, is the *sorted access* or sequential access to a sorted list. Formally we can say, that sorted access can be described by function $r_i(s_i(j))$. Using this type of access the grades of objects are obtained typically by proceeding through the list sequentially from the top. Let's assume that we have also a monotone aggregation function F, which combines grades of object x from lists $L_1, \ldots, L_m$. We denote the overall value of an object x as $S(x)$ and it is computed as $F(r_1(x), \ldots, r_m(x))$.

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

We will discuss two main cases. In the first case an algorithm will use both sorted and random accesses, and in the second case we will permit only sorted access. We will show one generalized algorithm for each case. Then we will try to adapt these algorithms to the user preference specifications. Note that all proposed methods can be easily modified for the use of skyline or monotone graded classification.

3 Generalized Threshold algorithm (TA) - uses both random and sorted access

First version of this algorithm was proposed by R. Fagin [7]. Generalized version was published in Gurský et al. [10, 11].

For each list L_i, let $u_i = r_i(s_i(z_i))$ be the value of the attribute of the last object seen under sorted access, where z_i is the position in the list L_i. Define the threshold value τ to be $F(u_1, \ldots, u_m)$. We assume that we have a monotone aggregation function F and the lists are sorted descend by their values from the best to the worst. During the execution of an algorithm we retrieve values from the lists form the greatest to the smallest, thus the threshold τ is the value, which none of still unseen objects can reach [7]. Hence when all objects in the top k list have their property values greater or equal than the threshold, then this top k list is the final and there is none unseen object with greater value. This property is very important to have the algorithm correct.

Let $z = (z_1, \ldots, z_m)$ be a vector, which assigns for each $i = 1, \ldots,$ m the position in list L_i last seen under sorted access. Let H be a heuristic that decides which list (or lists) should be accessed next under sorted access. Moreover, assume that H is such, that for all $j \leq m$ we have $H(z)_j = z_j$ or $H(z)_j = z_j+1$ and there is at least one $i \leq m$ such that $H(z)_i = z_i+1$. The set $\{i \leq m : H(z)_i = z_i + 1\}$ we call the set of candidate lists (or simply candidates) for the next sorted access.

The generalized Threshold algorithm is as follows:

0. Set z:=(0,...,0)
1. Set the heuristic H and do the sorted access in parallel to each of the sorted lists to all positions where $H(z)_i = z_i+1$. Put $z_i = H(z)_i$.

2. First check: Compute the threshold value τ. As soon as at least k objects have been seen whose grade is at least equal to τ , then go to step 5.
3. For every object x that was seen under sorted access in the step 2, do the random access to the other lists to find the grade $x_i = r_i(x)$ of object x in every list. Then compute the grade $S(x) = F(x_1, \ldots, x_m)$ of object x. If this grade is one of the k highest ones we have seen, then remember object x and its grade $S(x)$.
4. Second check: As soon as at least k objects have been seen whose grade is at least equal to τ , then go to step 5, 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 $\{(x, S(x)) : x \in Y\}$.

The easiest heuristic is the heuristic in Threshold algorithm [7]. This heuristic chooses all the lists as candidates, i.e. $H(z)_i = z_i{+}1$ for every i. For overview of other heuristics see [9, 10]. This algorithm is correct [7] for any heuristic and instance optimal for some of heuristics [7, 10, 11]. The instance optimality guarantee that for any data the algorithm do at most m^2 times more accesses then in the ideal case.

4 Three phased no random access (3P-NRA) algorithm - uses only sorted access

3P-NRA algorithm was firstly presented by P.Gurský in [13] and it is an improvement of NRA algorithm [7].

First of all we need to define worst and best value. Given an object x and subset $V(x) = \{i_1, \ldots, i_n\} \subseteq \{1, \ldots, m\}$ of known attributes of x, with values $x_{i_1}, \ldots, x_{i_n}$ for these fields, define $W_V(x)$ (or shortly $W(x)$ if V is known from context) to be minimal (*worst*) value of the aggregation function F for the object x. Because we assume that F is monotone aggregation function, we can compute its value by substituting for each missing attribute $i \in \{1, \ldots, m\} \backslash S$ the value 0. For example if $V(x) = \{1, \ldots, g\}$ then $W_V(x) = F(x_1, \ldots, x_g, 0, \ldots, 0)$.

Analogously we can define maximal (*best*) value of the aggregation function F for object x as $B_V(x)$ (or shortly $B(x)$ if V is known from context). Since we know that values in the lists are ordered descended we can substitute for each missing property the values along the vector z. For example if $V(x) = \{1, \ldots, g\}$ then $B_V(x) = F(x_1, \ldots, x_g, u_{g+1}, \ldots, u_m)$.

The real value of the object x is $W(x) \leq S(x) \leq B(x)$. Note that the unseen object (no attribute values are known) has $B(x) = \tau = F(u_1, \ldots, u_m)$ and $W(x) = F(0, \ldots, 0)$. On the other hand if we know all the values $W(x) = B(x) = S(x) = F(x_1, \ldots, x_m)$.

The 3P-NRA algorithm is as follows:

I. Descending with the threshold and the heuristic $H1$
   0. Set z:=(0,. . . ,0)
   1. Set the heuristic H1 and do the sorted access in parallel to each of the sorted lists to all positions where $H1(z)_i = z_i{+}1$. Put $z_i = H1(z)_i$.

2. For every object x seen under sorted access in the step 1, compute $W(x)$ and $B(x)$. If the object x is relevant, put x in the list T, that is the list of relevant objects ordered by *worst* value (an object x is relevant, if less than k objects was seen or $B(x)$ is grater than k-th biggest *worst* value in T). If the object x is not relevant remove it from T.
3. If we have at least k objects in T with greater *worst* value than τ go to phase II. otherwise go to step 1 of phase I.

II. Removing irrelevant objects
Compute *best* value for each object in T between the $(k+1)$-th and the last one. If an object is not relevant remove it from T. If $|T| = k$ return T otherwise go to phase III.

III. Descending with the heuristic $H2$
1. Set the heuristic $H2$ and do the sorted access in parallel to each of the sorted lists to all positions where $H2(z)_i = z_i{+}1$. Put $z_i = H2(z)_i$.
2. For every object x that was seen under sorted access in the step 1 of this phase do: If $x \notin T$ ignore it, otherwise compute $W(x)$ and $B(x)$. If the object x is relevant, move x to the right place in the list T. If the object x is not relevant remove it from T.
3. If $|T| = k$ return T
4. If by moving in T the k-th value of T was changed or the value of τ was decreased go to phase II, otherwise repeat phase III.

As heuristic H1 we can choose the heuristic from Threshold algorithm again. As heuristic H2 we can use heuristic *holes* [13], which chooses as candidates the lists with lowest number of known values in T. This algorithm is also correct [7] and instance optimal with the use of heuristic from Threshold algorithm.

5 Extensions

In all proposed extension we assume that the lists $L_1, \ldots, L_m$ are ordered by real values of properties from the smallest to the biggest, thus not from the best to the worst (it is not possible in general case). For example the distances from the centre of the city will be ordered from nearest to the most far. Next we assume that we have user fuzzy function for each property and it is one of 4 types like on figure 1. Let f_i be the fuzzy function for the list L_i. The overall fuzzy score of the object x will be $S_f(x) = F(f_1(x_1), \ldots, f_m(x_m))$. We will call $f_i(x_i)$ the fuzzy value of i-th property of the object x.

The main principle of both TA and 3P-NRA algorithms is to retrieve top k objects correctly without reading whole lists, thus using as low number of accesses as possible. Adding fuzzy functions, the situation is getting more complicated. Descending the lists by the real value causes that the threshold in previous algorithm does not guarantee the correctness any more. However the situation can be better when for some lists we have the fuzzy functions of type A. In this case such lists provide data from the best to the worst i.e. as it was in previous algorithms.

In the following we assume that lists $L_1, \ldots, L_a$ are all lists with fuzzy functions of type A, $L_{a+1}, \ldots, L_b$ are all lists with fuzzy functions of type B, and $L_{b+1}, \ldots, L_c$ and $L_{c+1}, \ldots, L_m$ are all lists with fuzzy functions of types C and D respectively.

5.1 Restricting sorted access

Bruno, Gravano, and Marian [14] discuss a scenario where it is not possible to access certain of the lists under sorted access. They did not consider fuzzy functions, but their solution can be correctly used in our case without any change. The only condition is that we have at least one list with the fuzzy function of type A, so we can do sorted access to this list. This solution is correct and instance optimal [7]. Algorithm is as follows.

1. Do sorted access in parallel to each list $L_1, \ldots, L_a$. For an object x seen under sorted access in some list, do random access as needed to the other lists to find the grade x_i of object x in every list L_i. Then compute the grade $S_f(x) = F(f_1(x_1), \ldots, f_m(x_m))$ of object x. If this grade is one of the k highest we have seen, then remember object x and its grade $S_f(x)$.
2. For each list L_i with $i \in \{a+1, \ldots, m\}$, let $u_i = 1$. As soon as at least k objects have been seen whose grade is at least equal to τ, then halt.
3. Let T be a set containing the k objects that have been seen with the highest grades. The output is then the graded set $\{(x, S_f(x)), x \in T\}$.

5.2 Reading whole list or waiting for a maximum

Now we propose the first solution for extension of 3P-NRA. We will read all the lists that have fuzzy functions of types B or D. Next we will read all the lists that have fuzzy functions of type C until they grow to the maximum fuzzy value. We can save accesses mainly to the lists with fuzzy function of type A and partially in the lists with type C. This solution can be helpful especially when we extend 3P-NRA algorithm. We will add a phase zero before the algorithm 3P-NRA:

0. Waiting for descending values
   0. For all i set $u_i = 1$ and compute the threshold value $\tau = F(u_1, \ldots, u_m) = F(1, \ldots, 1)$. In this phase u_i is fixed for all i because we need to keep the threshold to be the upper bound of all unseen object values.
   1. Choose one list L_i from $L_{a+1}, \ldots, L_b, L_{c+1}, \ldots, L_m$ and read whole list L_i by sorted accesses. If there is no such a list go to step 3. Put all objects to list T ordered by the *worst* value. Set $u_i = 0$ and compute new threshold. If any object seen is no more relevant (when its *best* value is smaller than *worst* value of the k-th object in T), remove it from T. After all, set $z_i = n$ i.e. to the last position in the list.
   2. If $|T| = k$ and τ is smaller or equal than the *worst* value of the k-th object in T, return T and halt. If there are unread lists with fuzzy function of type B or D and there are more than k relevant objects in T go to step 1. Otherwise go to step 3.

3. For each list L_i from $L_{b+1}, \ldots, L_c$ read the list L_i up to position where the fuzzy value of the property reach the maximum value i.e. value 1. If there is no such a list go to step 1 of phase I. After each list do the same check as in step 2.

It can be easily seen that adding the phase 0 before 3P-NRA solves our problem correctly. The main idea of this phase is to reach the best values in all lists after whose we have the same start situation as we had in the original 3P-NRA algorithm.

Theorem 1. *The last extension of the algorithm 3P-NRA is correct.*
Proof. The objects are removed only when they are not relevant. The question is: if an object become irrelevant, should it become relevant again? By other words if its *best* value is smaller than *worst* value of the k-th object in T should its *best* value be later greater? Since u_i is fixed to 1 for all i for current read list (it does not change by sorted access), it is larger or equal to the real value of the object. Moreover we assume that the aggregation function is monotone. Hence the best value of any object decreases only. Since the list T is ordered by *worst* values and *worst* values of all objects increase only, the *worst* value of the k-th object in T increases only. Considering both these facts, we can see that when the object become irrelevant it cannot become relevant anymore. The last thing to solve is the question if all possible objects are considered to be in top k. If at least one whole list is read in step 1 all objects are considered automatically. If we read the lists in phase 0 only in step 3 again all objects up to highest fuzzy value are read. The rest of values are read in other phases (I.-III.) and as it was shown in [13], these phases are correct. □

The phase 0 can also be put before algorithm TA. This algorithm works well, when we do not have the property with the fuzzy function of type A too.

5.3 Two ways descending

The next extension of both TA and 3P-NRA algorithms will cause the same performance in the case of each fuzzy function type as the algorithms TA and 3P-NRA in the original task. To reach such a performance, we need lightly upgrade the functionality of data sources. We will require:

- A source will provide two lists for sorted access - first will send objects with property values ordered from the biggest to the smallest (descending order) and second will send data from the smallest to the biggest (ascending order). It can be implemented for example as two pointers on the same ordered list - one goes from left to right and the second goes from right to left.
- Lists can start sending data from the specified value.

When we have such functionality we can easily simulate the source that sends data from the best to the worst. Moreover we guarantee that we do not need any reordering or any other computation on the side of source.

When we have a property with fuzzy function of type A or B we can easily simulate the "best-worst" source by choosing the suitable list of the source -

ascended or descended. In this case, one request for a sorted access from a central algorithm means to do one sorted access to the real source and computation of fuzzy value.

To simulate the source using the fuzzy function of type D, we can use both lists and start from the first record in each list. Thus we get the biggest value of the active domain of the given property from the first list and the smallest value of the same property from the second list. After the computation of fuzzy values of both retrieved values, we can send to the algorithm the greater one. After next request from the algorithm, we must do the sorted access to the list from which we sent the value last time and again compare fuzzy values computed from both lists.

Assume that from the top of the first list we will retrieve fuzzy (computed by fuzzy function) values $(o_1, 1.0), (o_2, 0.8), (o_3, 0.7), \ldots$ and from the second list fuzzy values $(o_4, 0.9), (o_5, 0.8), (o_6, 0.6), \ldots$. After first request we need to do sorted access to both lists and retrieve objects o_1 and o_4 with fuzzy values 1.0 and 0.9 respectively. 1.0 is greater than 0.9, so we send to the algorithm $(o_1, 1.0)$. After next request, we will do the sorted access to the first list and retrieve object o_2 with fuzzy value 0.8. We send greater $(o_4, 0.9)$, thus the next request will cause the sorted access to the second list. After receiving $(o_5, 0.8)$, we can randomly choose, which object has to be sent. If we choose o_2, the next sorted access will be to the first list. The objects o_5, o_3 and o_6 will be sent at the end.

The simulation of the source using the fuzzy function of type C needs also the second requirement - to start sending data from the specified value. If we want to send the values from the best to the worst, we need to start from the value with maximal fuzzy value. It means to start in the "middle" of the list to both ends, or also from the same specified value in both ordered lists. Now we are in the same situation as in the case of fuzzy function of type D and we can use the same combination procedure of two ordered lists.

As can be seen using this approach we can simulate the "best-worst" sources with the same number of accesses to the sources except one sorted access for each source with fuzzy function of type C or D. Thus we can use all known algorithms developed for the "best-worst" sources with the same good performace.

We use the two ways descending method in the tool $top - k\ aggregator$ in the project NAZOU[1]. The main task of this tool it to find top k job offers for a user.

6 Conclusion

In this paper we extended the model of distributed multifeature querying by adding user specification of preferences to properties values. Such a model allows better specification of the idea of good object using object properties. We propose the extensions of known algorithms to work over this model. Proposed solutions are needed especially in the cases when we cannot reorder the lists in provided

[1] http://nazou.atrip.sk

sources. Reordering is quite difficult when fuzzy functions come together with the query.

In the future work is the comparison of proposed algorithms over real data. In present we have implementation of the last extension. We can see from the design of the algorithms that it is the best, because it works as good as current well known algorithms over simplest model. Other algorithms should be useful, when there cannot be required functionality in the sources. On the other side the extensions work with individual sources, hence the approaches should be combined.

References

1. U.Güntzer, W.Balke, W.Kiessling *Optimizing Multi-Feature Queries for Image Databases*, proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000
2. U.Güntzer, W.Balke, W.Kiessling *Towards Efficient Multi-Feature Queries in Heterogeneous Environments*, proceedings of the IEEE International Conference on Information Technology: Coding and Computing (ITCC 2001), Las Vegas, USA, 2001
3. W.Balke, U.Güntzer, J. Zheng *Efficient Distributed Skylining for Web Information Systems*, proceedings of the 9th International Conference on Extending Database Technology (EDBT 2004), LNCS 2992, Heraklion, Crete, Greece, Springer, 2004
4. W.Balke, U.Güntzer *Multi-objective Query Processing for Database Systems*, proceedings of the 30th International Conference on Very Large Databases (VLDB 2004), Toronto, Canada, 2004
5. W.Balke, U.Güntzer *Efficient Skyline Queries under Weak Pareto Dominance*, proceedings of the IJCAI-05 Multidisciplinary Workshop on Advances in Preference Handling (PREFERENCE 2005), Edinburgh, UK, 2005
6. R.Fagin *Combining fuzzy information from multiple systems*, J. Comput. System Sci., 58:83-99, 1999
7. R.Fagin, A.Lotem, M.Naor *Optimal Aggregation Algorithms for Middleware*, proc. 20th ACM Symposium on Principles of Database Systems, pages 102-113, 2001
8. S.Börzsönyi, D.Kossmann, K.Stocker *The Skyline Operator*, ICDE 2001: 421-430, Heidelberg, Germany, 2001
9. M.Vomlelová, P.Vojtáš *Pravděpodobnostní pohled na víceatributové dotazy v distribuovaných systémech*, Proceedings of ITAT 2005, p. 167-175, 2005
10. P.Gurský, R.Lencses *Aspects of integration of ranked distributed data*, proc. Datakon , ISBN 80-210-3516-1, pages 221-230, 2004
11. P.Gurský, R.Lencses, P.Vojtáš *Algorithms for user dependent integration of ranked distributed information*, technical report, 2004
12. P.Gurský, T.Horváth *Dynamic search of relevant information*, Proceedings of Znalosti 2005, pages 194-201, 2005
13. P.Gurský *Algoritmy na vyhľadávanie najlepších k objektov bez priameho prístupu*, Proceedings of Znalosti 2006, pages 95-105, 2006
14. N. Bruno, L. Gravano, and A. Marian *Evaluating top-k queries over web-accessible databases*, proceedings of the 18th International Conference on Data Engineering. IEEE Computer Society, 2002.
15. Šuc, D. *Machine Reconstruction of Human Control Strategies*, Volume 99 of Frontiers in Artificial Intelligence and Applications. Amsterdam, IOS Press, 2003.