Dynamic search of relevant information[1]

Peter Gurský, Tomáš Horváth

Institute of Computer science, University of P. J. Šafárik,
Jesenná 5, 041 54 Košice
Slovak Republic
gursky@upjs.sk, thorvath@science.upjs.sk

Abstract. In this paper we introduce the method for the integration of ranked distributed data with use of dynamically learned monotone aggregation function. We use the method of monotone graded classification to learn the new aggregation functions in dependency of user preferences on returned objects. This method helps user to specify his/her requirements without any knowledge about the values of attributes of the objects and retrieve more relevant object consecutively. We show our method on an illustrative example.

Keywords: integration of distributed information, middleware, aggregation, generalized annotated programs

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. There are several algorithms in this area, which solve this problem. Fagin [2] introduced "Fagin's algorithm" which solve this problem first time. Fagin et al. [3] presented "threshold" algorithm. Nepal et al. [4] defined algorithm that is equivalent to threshold algorithm. Güntzer et al. [1] define "quick-combine" algorithm with heuristic rule that determines which sorted list should be preferred. None of them do support any learning of the aggregation function, which is used to combine the attributes.

In this paper we use the modification of the *x/Δx switch algorithm*, which was presented by Gurský et al. [11]. To learn the aggregation function, which depends on user preferences, we use the method of monotone graded classification presented by Horváth et al. [10]. For better understanding of our motivation, we introduce the following example.

This example concerns with hotel reservation system in Košice, Slovakia. We suppose the following query:

```
Find hotels in Košice that are far from the airport,
their cost is reasonable and their building is fine.     (1)
```

[1] This work was supported by the grant VEGA 1/0385/03

In order to process this query we need to specify a user's notions of being near to, cost reasonable and building is fine by the fuzzy sets for example. To retrieve and order hotels we also need an aggregation function, which compute the overall score of hotels according to values of all three attributes. It is possible, that each attribute is supported by the different server (web service).

Our idea of the middleware system, which supports these requirements, is as follows. The user specifies his/her fuzzy constants by the fuzzy sets and sent them to the web services. With the use of the fuzzy set, the web service can compute the rank of the supported attribute of each object and create the list of the objects ordered by the ranks. It is also possible that the web service does not support user rankings and use only its own (possibly hidden) ranking. In this case the web service has the only one ordered list. Now the user can specify, which attribute is more important for him/her and which is not. For example the user can preference the hotels that are cheap before the new ones. This is possible to do it by creating an aggregation function (i.e. a weighted mean). Otherwise, we can use the simple arithmetic mean. Now the system computes the top *k* objects using the *x/*Δ*x switch algorithm*. In the ideal case the user becomes exactly the object, which he/she was searching for. Otherwise the user can estimate each retrieved object using some scale (commonly 0…10 (from very bad to very good)). Based on these values the system computes the new aggregation function and retrieves new top *k* objects, which are more relevant than the previous ones.

The other idea of the middleware does not allow the user to now, which or how many attributes are used by the system. In the first step the user receives some objects from the domain. After that he/she can evaluate each object using the scale, than the system computes an aggregation function and retrieve the top *k* objects. In this case, the query from our example can look like: "*Find hotels in Košice that are good for me*".

The last idea can be the combination of the previous ones. The user could use, during the specification of his/her own aggregation function, only some of the known attributes. For example the user's query can be: "*Find hotels in Košice with the reasonable cost*".

The other areas in which the learning of aggregation function can be useful are information retrieval or multimedia databases.

In this paper we present such a system that can solve these problems.

In chapter 2 we describe the *x/*Δ*x switch algorithm*. In chapter 3 we discuss, which learning method can be suitable. In chapter 4 we present the *method of monotone graded classification*. Chapter 5 concludes our paper.

2 Aggregation

Assume we have a finite set of objects. Cardinality of this set is *N*. Every object *x* 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 attribute x_i. We can define two functions to access objects in lists. Let *x* be an object. Then $s_i(x)$ is the 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. The lists can be accessed in two ways. First one is *sorted* (sequential) *access*. Using this type of access the grades 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. The overall value of object *x* we denote as *S(x)* and it is computed as $F(s_1(x), \dots s_m(x))$.

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.

There are many algorithms in this area [1, 2, 3, 4, 11]. Gurský et al. [11] experimentally showed, that the most powerful is the *x/Δx switch algorithm*.

2.1 x/Δx switch algorithm

For each list L_i, let $u_i = s_i(r_i(z_i))$ be the grade of the attribute of the last object seen under sorted access. Define the *threshold value* τ to be $F(u_1, \dots, u_m)$. Because we assume that we have a monotone aggregation function *F* and the lists are sorted descend by their values, the threshold is the value, which none of still unseen objects can reach [2]. This enforces that when all objects in the top *k* have their values greater or equal to the threshold, then this top *k* list is final and there is none unseen object with greater value. This property is very important to have the algorithm correct.

Let $z = (z_1, \dots z_m)$ be a vector, which assigns each $i = 1, \dots m$ 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. 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 for the next sorted access.

In this algorithm we use three types of heuristics.

First type of heuristics (denote $\boldsymbol{H_1}$) do the parallel sorted access in each list. It means that for each $i \leq m$ holds $H(z_i) = z_i+1$. This heuristics was firstly presented by Fagin et al. [3] in “threshold” algorithm. This kind of heuristics is used only in the first phase of computation to retrieve the beginnings of the lists. Next two heuristics are used in the rest of computation.

The use of the $((\partial F/\partial x)*\Delta x)$ heuristics ($\boldsymbol{H_2}$) was first presented by Güntzer et al. [1] as a part of “quick-combine” algorithm. Let us look at the next unequation. For each object *x* in the final top *k* list must hold:

$$S(x) = F(s_1(x), .. s_m(x)) \geq \tau \qquad (2)$$

Hence, when we can say, that this unequation holds, we have the final top *k* list. Obviously, there are two ways to make (2) hold: to increase the left side or to decrease the right side. Heuristics H_2 tries to decrease τ as fast as possible. As a criterion which list is suitable in the next sorted access, we use the next equation:

$$\Delta_i = \left(\frac{\partial F}{\partial x_i} (s_1(r_1(z_1)), \dots, s_m(r_m(z_m))) \right)^{-} * (s_i(r_i(z_i - p)) - s_i(r_i(z_i)) \qquad (3)$$

The constant p is some suitable (small) natural number. Hence Δ_i is the multiplication of the partial derivative of aggregation function F from the left in the point $(s_1(r_1(z_1)),..., s_m(r_m(z_m)))$ and the expected change of values in p steps (Δx factor) of the i-th list. When we have Δ_i for each $i \leq m$, we can set the value of $H(z_i)$. Heuristics H_2 sets $H(z_i)=z_i+1$ if $\Delta_i = \max\{\Delta_j;\ j \leq m\}$. Otherwise $H(z_i)=z_i$. The only necessary condition we required from F is the continuity from the left.

The $((\partial F/\partial x)*x)$ heuristics (***H*** $_3$) is a variation of the last one. This heuristics was presented by Gursky et al.[11] at the first time. Instead of following the Δx factor, H_3 chooses an x-factor, which is the last seen value in the i-th list. The criterion for this heuristics is:

$$c_i = \left(\frac{\partial F}{\partial x_i}(s_1(r_1(z_1)),...,s_m(r_m(z_m))) \right)^{-} * (s_i(r_i(z_i))) \tag{4}$$

The criterion (4) computes the partial derivation of F from the left in the point $(s_1(r_1(z_1)),..., s_m(r_m(z_m)))$ and multiply it with value in the point $s_i(r_i(z_i))$ in L_i. This heuristics sets $H(z_i)=z_i+1$ if $\chi_i = \max\{\chi_j;\ j \leq m\}$. Otherwise $H(z_i)=z_i$. We need the continuity from the left for the function F again.

When the aggregation function is the simple weighted mean, the derivation used by these heuristics is a constant, more precise it is the weight of the attribute.

Now we can describe the *x/Δx switch algorithm*:

```
0. $z$:=(0,...,0), set the suitable small natural $p$.
1. if any $z_i<p$ then $H:=H_1$; otherwise if $H=H_1$ then $H:=H_2$; if $H=H_2$ then $H:=H_3$;
   and if $H=H_3$ then $H:=H_2$.
2. Do sorted access in parallel to each of the sorted lists to all positions where
   $H(z)_i = z_i+1$. Put $z_i = H(z)_i$ .
3. First control: Compute the threshold value $\tau$. As soon as at least $k$ objects
   have been seen whose grade is at least equal to $\tau$ , then go to step 6.
4. With the object $x$ that was 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 $k$ objects and their grades need to be
   remembered at any time).
5. Second control: As soon as at least $k$ objects have been seen whose grade is at
   least equal to $\tau$ , then go to step 6, otherwise go to step 1.
6. Let $Y$ be a set containing the $k$ objects that have been seen with the highest
   grades. The output is then the graded set $\{(x, S(x)) : x \in Y\}$ .
```

Algorithm 1. x/Δx switch algorithm

The main idea of this algorithm is to try to keep the left hand side of (2) big (using the heuristic H_3) and to decrease the right hand side of (2) (heuristic H_2) simultaneously.

3 Aggregation versus classification

There are some problems in case of changing the aggregation function during the search. One of them is that the implementation of the computed aggregation function to the algorithm is complicated (the evaluation of the partial derivation). The second is that sometimes it is hard to compute the aggregation function. That is why we use instead of computing of aggregation function the results of classification methods. Its advantage is that the results are interpreted as a table, so it is easy to use.

Now we have little problems how to change the *x/Δx switch algorithm* to compute the overall score of each object when we do not have any aggregation function but the classification of objects in dependence of the attribute values. We need to solve two things. Firstly we need to simulate the computation of aggregation function and secondly we need to define how to compute the criteria of heuristics H_2 and H_3.

The meaning of any classification rule is as follows: When the attributes of object x have the values greater or equal to relevant values on the right side of the rule then the overall value of x is at least the same as on the left side of the rule. Hence during the simulation of computation of aggregation function we can simply test the validity of requirements of the rules from the strongest one to weaker rules. When we find the rule that holds, we can say that the overall value of the object is the value on the left side of the rule. Since we test the sorted rules, we always rank the object with the highest possible value.

The problem of computing the criteria Δ_i and χ_i of heuristics H_2 and H_3 is at first sight in the interpretation of the partial derivation. In the case of classifications we do not need any computation. We must to remind the first motivation of both heuristics. H_2 tries to decrease the threshold value and H_3 tries to keep the values of new objects high. Hence in the first case we can do the sorted access to the list, in which the last seen value is the nearest to the lower bound of the class (values on the right side of the rules). The second heuristic prefers the list of which the last seen value is the most far from all lower bounds of class. The H_2 heuristic can refined by parameter p when we look back in each attribute to analyze the speed of decreasing to the lower bounds and overall criterion for this heuristic can be some combination of distances and the speed. The problem of this combination is the object of our next research.

In figure 2 we can see the user classification of some concrete hotels. The result of classification method can be interpreted as rules written in following table:

```
y(A,0.9) :-   x1(A,0.7), x2(A,0.8).
y(A,0.8) :-   x1(A,0.9), x2(A,0.6).
y(A,0.8) :-   x1(A,0.6), x2(A,0.7).
y(A,0.8) :-              x2(A,0.9).
y(A,0.7) :-              x2(A,0.8).
y(A,0.7) :-   x1(A,0.6), x2(A,0.6).
y(A,0.7) :-   x1(A,0.3), x2(A,0.7).
y(A, 0.7) :-  x1(A,0.7), x2(A,0.5).
y(A, 0.6) :-  x1(A,0.6), x2(A,0.3).
y(A, 0.6) :-  x1(A,0.3), x2(A,0.6).
y(A, 0.6) :-  x1(A,0.4), x2(A,0.5).
```

```
y(A, 0.6) :-             x2(A,0.7).
y(A, 0.5) :-  x1(A,0.1), x2(A,0.5).
y(A, 0.5) :-             x2(A,0.6).
y(A, 0.5) :-  x1(A,0.3), x2(A,0.3).
y(A, 0.5) :-  x1(A,0.4), x2(A,0.2).
y(A, 0.4) :-  x1(A,0.2), x2(A,0.2).
y(A, 0.4) :-  x1(A,0.4), x2(A,0.1).
y(A, 0.4) :-             x2(A,0.4).
y(A, 0.3) :-  x1(A,0.3).
y(A, 0.3) :-             x2(A,0.1).
```

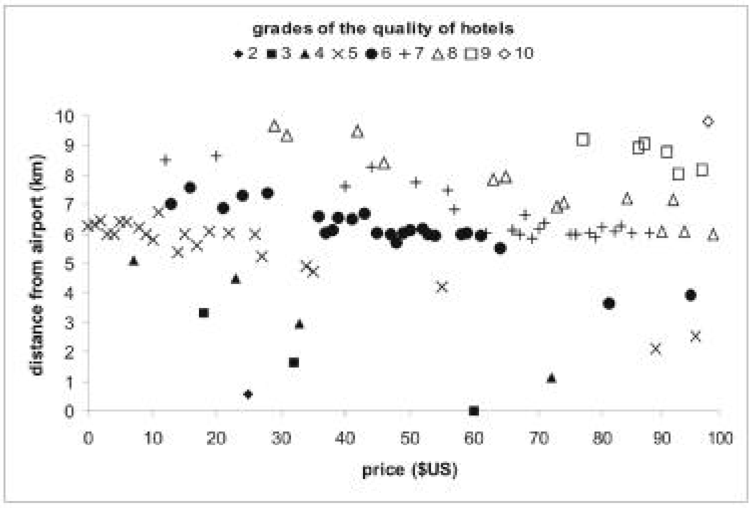

Figure 1. Illustrative example of users overall preference of (100) hotels. The grades 2, 3,..., 10 means the classes of "the worst", "bad", ..., "the best" hotels according to users criterion. User did not rank any hotel by the grades 0 or 1.

This table is graphically represented in figure 2. It can be easily seen that the computation of the overall value or computing the heuristics is easy to solve. Using this classification we are able to retrieve more relevant objects to user requirements.

The use of some classification method in the *x/Δx switch algorithm* is conditioned only by monotone classification, but selected method is suitable.

4 Learning the classification

Main point of interest of this chapter is to learn the (user dependent) classification (ranking, grading, …), which gives and overal score to a graded fulfilment of all user's requirements. This appears also in mulcriterial and/or multiuser decision making and also in graded classification (where the monotonicity of dependence can be more problematic).

For our purposes it is convenient to induce rules of the form

IF body_attribute$_1$ ≥ (≤) grade$_1$ AND ... AND body_attribute$_n$ ≥ (≤) grade$_n$ THEN head_attribute ≥ (≤) grade$_H$	(5)

where grade$_H$ = @(grade$_1$, ..., grade$_n$), @ is an n-ary aggregation operator.

These types of rules preserve the monotonicity (both in the directions down and up depending on the directions of the monotonicity of attributes).

Our method of learning the (monotone graded) classification [10] is based on multiple use of classical Inductive logic programming system ALEPH [9] with additional monotonicity axioms in the background knowledge.

4.1 Inductive logic programming ILP

Inductive logic programming ILP is a data mining method which combines inductive learning with predicate logic. The use of the language of predicate logic makes this method effective, especially in multi-relational data mining.

In ILP, given is a set of examples, i.e., tuples that belong to the target relation p (positive examples) and tuples that do not belong to p (negative examples). Given are also background relations (or background predicates) q_i that constitute the background knowledge and can be used in the learned definition of p. Finally, a hypothesis language, specifying syntactic restrictions on the definitions of p is also given (either explicitly or implicitly). The task is to find a definition of the target relation p that is consistent and complete. Informally, it has to explain all positive and none of the negative tuples.

More formally, given is a set of examples $E = E^+ \cup E^-$, where E^+ contains positive and E^- negative examples, and background knowledge B. The task is to find a hypothesis H such that the following holds:

$$\begin{array}{l} \forall e \in E^+ : B \wedge H \models e \text{ (H is complete)} \\ \forall e \in E^- : B \wedge H \not\models e \text{ (H is consistent).} \end{array} \tag{6}$$

This setting, introduced by Muggleton ([8], see also [7]), is also called learning from entailment.

4.2 Monotone graded classification

In the *monotone graded classification task* the instances of the target/head attribute E_G (set of graded examples) and instances of body/background attributes B_G (graded background knowledge) of input relation are given. Our task is to find a hypothesys (set of rules "head ← body") H_G such that for every e_1, e_2 from E_G the following holds:

$$\begin{array}{l} E_G(e_1) < E_G(e_2) \quad \text{implies} \quad (H_G \cup B_G)(e_1) \leq (H_G \cup B_G)(e_2) \\ E_G(e_1) > E_G(e_2) \quad \text{implies} \quad (H_G \cup B_G)(e_1) \geq (H_G \cup B_G)(e_2) \end{array} \tag{2}$$

That is, our classification H_G according to B_G does not contradict the original classification E_G, or in other words, H_G (according to B_G) does not declare an object e_1 strictly better than e_2 if in E_G is just the opposite, i.e. e_2 strictly better than e_1.

We choose the system ALEPH because it can work with rules in the background knowledge (it is useful for introducing monotonicity axioms).

```
Input: $B_G$ and $E_G$ of the input relation
Output : $H_G$
{
   $G_B \leftarrow \{\{\alpha_{k1}, ..., \alpha_{km} \mid m \in N^+ \text{ and } (i \neq j \rightarrow \alpha_i \neq \alpha_j)\} \mid k \leq$ number of predicates in $B_G\}$,
   $G_E \leftarrow \{\alpha_1, ..., \alpha_n \mid n \in N^+ \text{ and } (i < j \rightarrow \alpha_i < \alpha_j)\}$,
   DISCRETIZE($B_G$, $G_B$)
   c($B_G$)   $\leftarrow$ CONVERT($B_G$)
   cm($B_G$) $\leftarrow$ ADD_MONOTONICITY_AXIOMS(c($B_G$), $G_B$)
   $H_G$     $\leftarrow \emptyset$
   for every $\alpha \in G_E \setminus \{\alpha_1\}$ do
   {
      $E_\alpha^+$  $\leftarrow \{e \in E_G: E_G(e) \geq \alpha\}$
      $E_\alpha^-$  $\leftarrow \{e \in E_G: E_G(e) < \alpha\}$
      $H_\alpha$  $\leftarrow$ ALEPH($E_\alpha^+$, $E_\alpha^-$, cm($B_G$))
      $H_G$  $\leftarrow H_G \cup H_\alpha$
   }
}
```

Algorithm 2. Rule induction for monotone graded classification task

On the beginning we find out the grades G_E of examples and the grades G_B of graded background knowledge attributes. Notice, that B_G can contain attributes with non discrete domain (real numbers) - we can understand these like (monotone) graded attributes with large number of grades. Because ILP systems understands numbers syntactically, for attributes with non discrete domain we apply the DISCRETIZE procedure (E.g. the values for the attribute price from the interval <0; 100> we discretize to 10 classes: α_{price1}= <0;10), α_{price2}=<10;20), ..., $\alpha_{price10}$=<90;100> (e.g. the class α_{price2} represents prices at least 10). Notice, that the finest discretization implies the better quality of classification, but increase the complexity of learning.

From the input relation hotel we have *E_G = hotelgrade(hotel_id, hotel_grade)* and *B_G = (hotel_id, price, distance)*. If we want to get rules of the form (1) we must separate each attribute from B_G. In our case CONVERT(B_G)={*price(hotel_id,price$_{hotel\_id}$), distance(hote_id,distance$_{hotel\_id}$)*} = c(B_G).

The monotonicity we define as follows: for graded body predicates (in our case we have just these) we add to the c(B_G) the rules *price(X,C) ← leg$_{price}$(C,D),price(X,D)* and *distance(X,C)← leq$_{distance}$(C,D), distance(X,D)*. Main idea is that distance(X,D) should fire the rule where distance(X,C) is in body too, because fulfilment in degree D is stronger than in degree C. *leq$_{attribute}$* are the prolog definitions of the operation "$C \leq_a D$" what means that the value C of the attribute a is less than or equal to the value D of this attribute (e.g. the first rule means that if the hotel costs at least D$ it also costs at least C$ what is "less than or equal to" D). We add to these monotonicity axioms the definitions of the predicates *leq$_{attribute}$* for every graded attribute according to their grades (in our case we add *leq$_{price}$(1,2), leq$_{price}$(2,3), ..., leq$_{price}$(9,10)* and *leq$_{distance}$(1,2), leq$_{distance}$(2,3), ..., leq$_{distance}$(9,10))*. This enriched c(B_G) we call cm(B_G) = ADD_MONOTONICITY_AXIOMS(c(B_G), G_B).

In our case the body attribute values contribute positively to higher classification in the natural ordering of the domain but in many cases these dependencies are not obvious. For checking these dependencies we can use several techniques – statistical methods, methods for qualitative learning as QUIN [6], etc. If the body attribute values contribute negatively to higher classification, instead *$leq_{attribute}$* we use *$geq_{attribute}$* (definition of "≥") with relevant definitions of these predicates, e.g. *$geq_{attribute}(10,9)$, ..., $geq_{attribute}(2,1)$.*

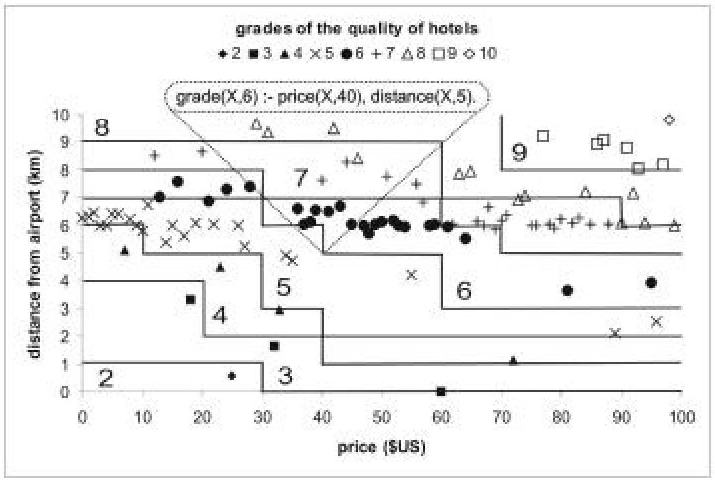

Figure 2. Result of the graded classification method with illustration of one rule

An example of an induced rule: *grade(X,6) :- price(X, 40), distance(x,5).* It means: IF price ≥ 40 $ AND distance ≥ 5 km THEN grade of the hotel ≥ 6.

These rules have the expected form and they preserve the monotonicity (the lowest class contains all hotels, the better just a few of them).

5 Conclusion

In this paper we proposed the method of searching of relevant information from distributed sources. The user is able to do his classification of retrieved objects to specify his/her requirements easily.

We integrated the modification of the *x/Δx switch algorithm* and the *method of monotone graded classification*. First of them use the classification from the second one instead of the aggregation function.

In the future work we want to implement this method and test it on the real data.

References

1. Güntzer, U., Balke, W., Kiessling, W.: Optimizing Multi-Feature Queries for Image Databases, Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000
2. Fagin, R.: Combining fuzzy information from multiple systems. J. Comput. System Sci., 58:83-99, 1999
3. Fagin, R., Lotem, A., Naor, M.: Optimal Aggregation Algorithms for Middleware. In Proc. 20th ACM Symposium on Principles of Database Systems, pages 102-113, 2001
4. Nepal, S., Ramakrishna, M. V.: Query processing issues in image (multimedia) databases. In Proc. 15th International Conference on Data Engineering, pages 22-29, 1999
5. 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, 2001
6. Bratko, I., Šuc., D.: Learning qualitative models. AI Magazine 24, pages 107-119, 2003
7. Džecroski, S., Lavrač, N.: An introduction to inductive logic programming. Relational data mining, S. Džeroski, N. Lavrač eds. Springer, 48-73, 2001
8. Muggleton, S.: Inductive logic programming. New Gen Comp 8, 295-318, 1991
9. Srinavasan, A.: The Aleph Manual. Tech. Rep. Comp. Lab. Oxford Univ. 2000 at http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html
10. Vojtáš, P., Horváth, T., Krajči, S., Lencses, R.: An ILP model for a monotone graded classification problem. Kybernetika 40, 317-332, 2004
11. Gurský, P., Lencses, R.: Aspects of integration of ranked distributed data. In proc. Datakon, 2004
12. Gurský, P., Lencses, R., Vojtáš, P.: Algorithms for user dependent integrationof ranked distributed information, technical report, 2004
13. Naito, E., Ozawa, J., Hayashi, I., Wakami, N.: A proposal of a fuzzy connective with learning function and query networks for fuzzy retrieval systems. In Fuzziness in database management systems. P. Bosc and J. Kacprzyk eds. Physica Verlag, 345-364, 1995