Top-k aggregator

Peter Gurský and Martin Šumák

UPJŠ, Košice, Slovakia
`peter.gursky@upjs.sk, sumi@netkosice.sk`
`http://klud.ics.upjs.sk/~gursky/, http://sumi.netkosice.sk/`

Abstract. In this paper we describe a tool named top-k aggregator. Using this tool we can retrieve some best objects with respect to user requirements. User can say which values of properties of offers (e.g. salary, education requirement, place, working hours per week) are interesting for him or her, how much and also which are not interesting. Another form of user specification can be made by evaluation of some objects in a training set. Individual properties can be, in some case, stored on different sources. We need to have an algorithm, that can return user dependent top k objects with small number of accesses to the sources, thus without retrieving all the sources. We can use this tool also for aggregation of different forms of searching.

1 Introduction and motivation

Top-k aggregator is a tool to aggregate potentially distributed properties of objects to find best objects to users trying to use as few accesses to sources as possible. The reason of which object is better than the other, is based on the properties of the objects. We implemented this tool for project NAZOU[1], where the objects are job offers.

Let us consider the following example. User is interested in good paid job, in manager position, he is good at languages and he want to travel but no a lot. So we will consider these four properties of objects. User can specify for each property the ordering of real values (he can say which real values are best - low, high or middle) by fuzzy functions. User can also specify the importance of each property by monotone aggregation function. The aggregation function has often the form of weighted sum e.g. the function

$$\frac{5 * good\_paid + 2 * middle\_travelling}{6} \tag{1}$$

specify higher preference for a salary. Top-k aggregator collect all these information, get essential data from sources and retrieve best objects. This process is schematically represented on figure 1.

[1] NAZOU = "tools for acquisition, organization and maintenance of knowledge in an environment of heterogeneous information resources", homepage: http://nazou.fiit.stuba.sk/

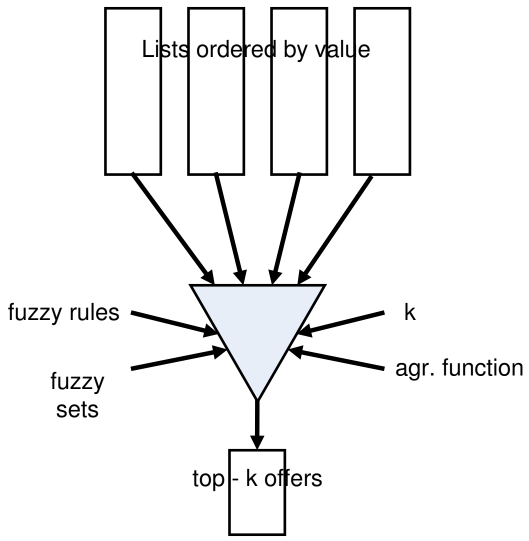

Fig. 1. Scheme of top-k aggregator

Useful condition is to assume, that each property is provided by a possibly remote source. 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.

We are interested in properties, that can be naturally ordered. This is important feature to get the power of this tool i.e. minimize the number of accesses. The first type of properties, we are interested in, are boolean or yes/no properties (e.g. work at home). The second type are properties, that are graded in some way to finite number of classes. Typical properties are: level of education, level of language required, amount of travelling etc. The third type are real or integer numbers for example: salary or hours per week. The last type of properties is text. We can use this kind of attribute to reduce searching space (user could search only IT jobs), especially, when such a property is organized in some hierarchical structure e.g. ontology.

When we find and specify all useful properties we can specify which properties are important for user and also how much. First approach to this problem is to use a mentioned monotone aggregation function. There are many solutions to find best objects using of aggregation function. Ronald Fagin in 1996 introduced "Fagin's algorithm", which solves this problem first time in [3]. Fagin et al. [4] 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 [6]. M. Vomlelová and P. Vojtáš [5]

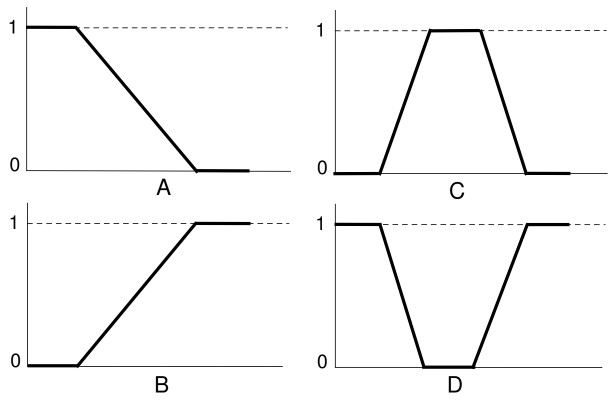

Fig. 2. Fuzzy functions

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 [4] 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 [9] with a new heuristic holes. All three approaches use only sorted access.

The random access has one big disadvantage. Each random access requires searching of the value of concrete object. In the case of sorted access the results are prepared immediately (values can be preordered to the several orderings (e.g in Fuzzy RDF) before user come) and, moreover, data can be sent in blocks. We can also use one elegant solution, in which we do not need to prepare any orderings (see chapter 2). Using the sorted access only, allows on input also streams from several programs.

There are also other approaches to specify preferences than the aggregation function. P. Gurský and T. Horváth in [8] 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.

For each property we consider, user can specify an ordering i.e. specify, which values of a property are better than the other. We will consider four types of fuzzy functions (see Figure 2). On the axis x, there are property values e.g.

manager level or salary. 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 "manager level". If we not prefer manager positions, our fuzzy function will look like the fuzzy function A on the figure 2. Fuzzy function B means that we prefer hi manager positions, fuzzy function C means our preference for middle manager positions. Fuzzy function D means that we prefer hi manager positions or not manager positions (very unstandard for this property). 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. Algorithms working with fuzzy functions was presented in [12].

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 [11].

Current version of Top-k aggregator tool allows to use all metioned heuristics and algorithms except probabilistic heuristic and combined algorithm.

The rest of the paper is organized as follows. In chapter 2 we describe main priciples of top-k aggregator on a core of 3P-NRA algorithm . In chapter 3 we write something about implementation. Chapter 4 concludes our paper and suggests a future research.

2 Model and 3P-NRA algorithm

In this chapter we show the core functionality of top-k aggregator by description of a one of possible algorithms we can use.

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 $L_1, \ldots, L_m$. Objects in list L_i are ordered descending by values of attribute x_i. 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. We can formally define the sorted access by function $r_i(s_i(j))$. Using sorted acccess 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 accesses as possible.

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 [4]. 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, ..., 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.

Next we need to define *worst* and *best* value of an object. 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 was firstly presented in [9] and works 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 easiest heuristic from Threshold algorithm [4]. 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 [5, 6]. As heuristic H2 we can use heuristic *holes* [9], which chooses as candidates the lists with lowest number of known values in T. This algorithm is correct [4] and instance optimal with the use of heuristic from Threshold algorithm. The instance optimality guarantee that for any data the algorithm do at most m^2 times more accesses then in the ideal case.

3 Implementation

The program can retrieve the best k objects on distributed data with respect to user requirements. It can make this retrieving by using six defferent algorithms. Each algorithm is implemented in its separate class as it is shown in the figure 3. The structure of their classes is arranged to allow the implementation of another algorithm without any changes in other parts of the program. It is advantageous because each algorithm respects defferent restrictions, which are of concern accesses to the data. The choice of the best algorithm depends on the way how the data is stored. The algorithms use the heuristics to maximize effeciency of the evaluation. Meanwhile the program has implemented seven heuristics. Each heuristic is implemented in its separate class as it is shown in the figure 3. Relations of dependencies among classes of algorithms and heuristics are shown in the figure 3 only on abstract level. It is so, because any algorithm can use any heuristic. The structure of the classes allows to implement another heuristic, usable for any algorithm again, without any changes in other parts of the program implementation. As it was mentioned, the lists which represent the data for retrieving the best k objects, can be stored in different data structures and on the different remote computers or repositories. The evaluation of the algorithms is independent to the method to store the lists. The algorithms work only by using random and sorted accesses to the lists. Thus, it is necessary to ensure uniformity of the behaviour of these lists whereby lower layer. This layer has to implement random and sorted access to these lists. This layer is in the program represented by class Access and subclasses of class SourceGeneral. The structure of these classes is shown in figure 3 too. Class Access provides

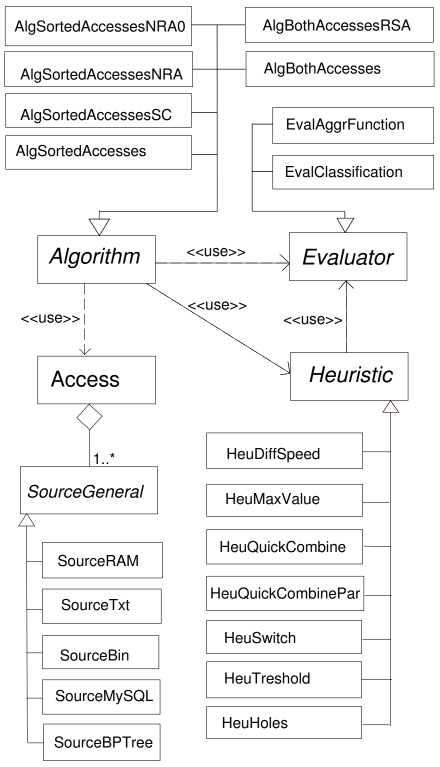

Fig. 3. Class diagram

for algorithms unified random and sorted accesses to the lists. Specific subclass of class SourceGeneral represents the specific implementation of the random and sorted access to the list stored in specific data structure. For example class SourceMySQL implements random and sorted access to the list stored in MySQL database. The program has implemented accesses to the lists stored in binary files, text files, MySQL database, B+ trees and by arrays in RAM. All the implementations of the accesses to the lists respect the same order of objects in the list for sorted access as it was defined by fuzzy function obtained from the user. It is possible to implement accesses to another data structures. Thus, class Access represents the aggregation of objects of different subclasses of the class SourceGeneral. The information about data structures and location of the individual lists is provided for the class Access by configuration XML file.

The implementation of B+ trees as new way to store the data in the list is the latest upgrade of the program. Owing to the fast random accesses to this data, much faster evaluation using B+ trees is expected. Random access to the other mentioned data structures except arrays in RAM was slow (MySQL database) or impossible at all (binary files, text files). It is not suitable or possible to store

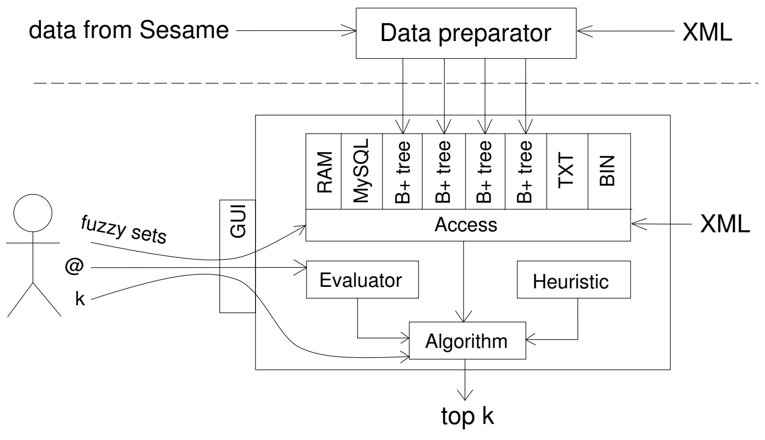

Fig. 4. Program schema

all the data in the list in RAM. Thus, necessity of the fast random access to the list which is not all stored in RAM lead to the implementation of the B+ tree. It is also sometime possible to write the data from other lists stored by other way (MySQL database, binary files, text files) to the B+ trees to minimize the time of the evaluation. B+ trees also provide new possibilities for preparation of the data which is stored by RDF in the Sesame database.

Query language SeRQL provided by the Sesame database doesn't support the ordering of data by a value. Thus, it is not practicable to do sorted access to the data stored in Sesame database. Therefore, it is necessary to prepare all the data stored in the Sesame by special way. This preparation means writing all the data from the Sesame to the other data structures which provide sorted access (MySQL database, binary files, text files, B+ trees). Data Preparator has been developed for this preparation. Data Preparator creates the lists stored in MySQL database, binary files, text files and B+ trees according the another configuration XML file. Performing of this preparation is independent to the program interaction iniciated by the user. This preparation is performed only once in a while, usually when the data in the Sesame was changed. Collaboration between the program components is shown in the figure 4. The part above the dashed line represents the preparation of the data from the Sesame. The part below the dashed line represents the evaluation iniciated by the user, who determines all the required information and then obtains the best k objects.

Evaluation of the overall value of an object is provided for the algorithms by class named Evaluator. The program has implemented two ways for this evaluation by aggregation function, and by rules of monotone graded classification acquired from the training set of user evaluated objects. This acquirement

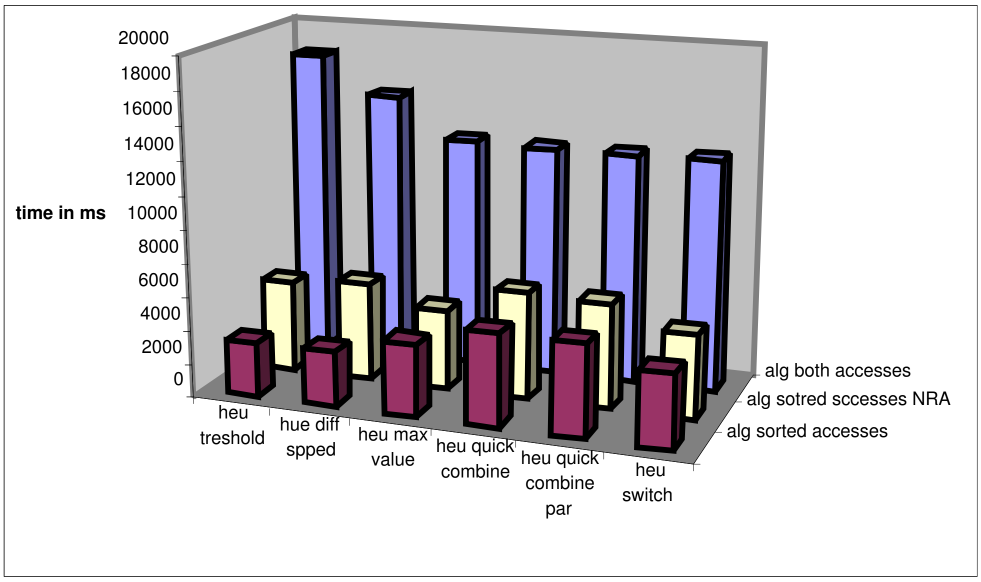

Fig. 5. Time of evaluation over MySQL

in project NAZOU is provided by the tool IGAP.The structure of the classes shown in the figure 3 allows to implement another ways of evaluation of the overall value of the objects whithout any change of other parts of the program implementation.

Figure 5 shows the time of the evaluation of three different algorithms in combination with six heuristics at the retrieving 10 best objects on data stored by MySQL database. Used test data contained 10 000 objects evaluated according to five properties. Slow random accesses induced that the algorithm both accesses which uses random accesses as much as possible is very slow and really unusable. This problem is supposed to be resolved by using fast random accesses to the data stored by B+ trees. It is expected that the algorithm both accesses will be the fastest in comparison with the others.

4 Future research

Main future work includes deeper tests of all possible parts of the system, especially B+ trees. We plan also implement other known heuristics and algorithms and compare them. Our next improvement relates with B+ trees again. One of good features of B+ trees are easy insertions and deletions of records. We can use them in Data Preparator to work only with newly added/deleted objects. Other possible upgrade will come with a new version of Sesame or other RDF database that supports ”ORDER BY” in selects.

Acknowledgements. This work was partially supported by the Slovak State Programme of Research and Development Establishing of Information Society under the contract No. 1025/04.

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. R.Fagin *Combining fuzzy information from multiple systems*, J. Comput. System Sci., 58:83-99, 1999
4. R.Fagin, A.Lotem, M.Naor *Optimal Aggregation Algorithms for Middleware*, proc. 20th ACM Symposium on Principles of Database Systems, pages 102-113, 2001
5. M.Vomlelová, P.Vojtáš *Pravděpodobnostní pohled na víceatributové dotazy v distribuovaných systémech*, Proceedings of ITAT 2005, p. 167-175, 2005
6. P.Gurský, R.Lencses *Aspects of integration of ranked distributed data*, proc. Datakon , ISBN 80-210-3516-1, pages 221-230, 2004
7. P.Gurský, R.Lencses, P.Vojtáš *Algorithms for user dependent integration of ranked distributed information*, technical report, 2004
8. P.Gurský, T.Horváth *Dynamic search of relevant information*, Proceedings of Znalosti 2005, pages 194-201, 2005
9. P.Gurský *Algoritmy na vyhľadávanie najlepších k objektov bez priameho prístupu*, Proceedings of Znalosti 2006, pages 95-105, 2006
10. 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.
11. Šuc, D. *Machine Reconstruction of Human Control Strategies*, Volume 99 of Frontiers in Artificial Intelligence and Applications. Amsterdam, IOS Press, 2003.
12. P.Gurský *Towards better semantics in the multifeature querying*, Proceedings of Dateso 2006, ISBN 80-248-1025-5, pages 63-73, 2006
13. P.Návrat, M.Bieliková, V. Rozinajová: *Methods and Tools for Acquiring and Presenting Information and Knowledge in the Web*. In: CompSysTech 2005, B. Rachev, A. Smrikarov (Eds.), Varna, Bulgaria, June 2005. pp. IIIB.7.1-IIIB.7.6.