Combination of TA- and MD-algorithm for Efficient Solving of Top-K Problem according to User's Preferences

Matúš Ondreička and Jaroslav Pokorný

Department of Software Engineering, Faculty of Mathematics and Physics,
Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic
matus.ondreicka@mff.cuni.cz,jaroslav.pokorny@mff.cuni.cz

Abstract. In this article we focus on efficient solving of searching the best K objects in more attributes according to user's preferences. Local preferences are modelled with one of four types of fuzzy function. Global preferences are modelled concurrently with an aggregation function. We focused on searching the best K objects according to various user's preferences without accessing all objects. Therefore we deal with the use of TA-algorithm and MD-algorithm. Because of local preferences we used B^+-trees during computing of Fagin's TA-algorithm. For searching the best K objects MD-algorithm uses multidimensional B-tree, which is also composed of B^+-trees. We developed an MXT-algorithm and a new data structure, in which MXT-algorithm can effectively find the best K objects by user's preferences without accessing all the objects. We show that MXT-algorithm in some cases achieves better results in the number of accessed objects than TA-algorithm and MD-algorithm.

1 Introduction and Motivation

Nowdays, huge amounts of data in various web systems grow exponentially. Users try to find various objects in this data, such as laptops, cars, apartments, holidays, etc. These objects have various attributes. According to the values of attributes, users search for the best objects for them. However, each user prefers objects with other values of attributes [7][6].

Most users look for only a few objects that are best suited to their preferences. Therefore, it is advantageous according to the user's preferences directly to find the best K objects. Making it possible to find the best K objects from the set of objects X, it must be possible to judge which objects are better or worse for a user U. Every user prefers objects with own preferences, which are modelled locally by means of a fuzzy function and globally by means of an aggregation function [9]. In this paper we assume that the set of objects of the same type is stored in one data structure. The problem of searching the best K objects according to values of different attributes in the same time is indicated as a *top-K problem* [4][7].

In the next, text we describle using B^+-tree [2] for sorting objects acording to fuzzy function for supporting the local preferences [1][3][10].

K. Richta, J. Pokorný, V. Snášel (Eds.): Dateso 2009, pp. 142–153, ISBN 978-80-01-04323-3.

We deal also with methods and data structures for effective solution of top-K problem [1]. We discuss the problem of finding the best K objects without accessing all the objects via Fagin's TA-algorithm [4] and MD-algorithm [1]. For application of local preferences TA- and MD-algorithm are using data structures based on B^+-trees [1]. These structures are independent from user's preferences. Moreover, it is possible to update these structures easily and quickly.

We developed a MXT-algorithm and a new data structure based on B^+-trees, in which MXT-algorithm can effectively find the best K objects by user's preferences without accessing all the objects. We will show advantages of the new MXT-algorithm and a comparison with the results of other algorithms. We show that MXT-algorithm in some cases achieves better results in the number of accessed objects than TA- and MD-algorithm.

A background concerning modelling user preferences is contained in Section 2. Section 3 presents use of B^+-tree for approaching the top-K problem. Sections 4 and 5 are devoted to explaining principles of Fagin's TA-algorithm and MD-algorithm. In Section 6 we describe our new MXT-algorithm and new data structure, which is used during computation of MXT-algorithm. Section 7 presents the results of the tests and compares MD-algorithm with Fagin's algorithms. Finally, Section 8 provides some suggestions for a future work.

2 User's preferences

In this article, we suppose a set of objects X with m attributes $A_1, ..., A_m$. Every object $x \in X$ has m values $a_i^x, ..., a_m^x$ of corresponding attributes. When searching for the best K objects, a user U chooses user's preferences, which determine suitability of the object $x \in X$ in dependence with its m values of attributes $a_1^x, ..., a_m^x$ for user U. In accordance to user U preferences, it is possible to find the best K object for user U. We are using a model of user preferences, which is based on concatenation of local preferences and global preferences. At first, user U decides how he prefers objects according to each of attributes A_i, $i = 1, ..., m$, and then chooses a relationship between these attributes.

Local preferences reflect how the object is preferred according to only one attribute. In this work, we used a mapping f_i: $dom(A_i) \rightarrow [0, 1]$. In general, we differentiate two possible attribute types.

Nominal attribute has a finite range of possible values, usually strings. For a nominal attribute A_i user U has to rate each value of the attribute. For example, it is mark of some products. Local preference of user U for the nominal attribute A_i is represented as mapping $f_i^U(x)$: $a_i^x \rightarrow [0, 1]$, where $i = 1, ..., m$.

Ordinal attribute has some natural value ordering, other than lexical ordering. Typical examples are integer numbers. Domain of ordinal attribute is subset of continuous interval. The local preference of user U for the ordinal attribute A_i is represented by *fuzzy function* [9]. We denoted it also as $f_i^U(x)$ the user's fuzzy function for i-th attribute. In this case, fuzzy functions have a scheme $f_i^U(x)$: $a_i^x \rightarrow [0, 1]$, where $i = 1, ..., m$.

For the worst object $x \in X$, $f_i^U(x) = 0$ holds and for the best one, $f_i^U(x) = 1$. Comparing $f_i^U(x_1)$ and $f_i^U(x_2)$ of two objects x_1 and x_2, it is possible to decide which of them is suitable for the user U according to the value of i-th attribute. When $f_i^U(x_1) > f_i^U(x_2)$, then x_1 is more suitable. When $f_i^U(x_1) = f_i^U(x_2)$, then x_1 and x_2 are equally suitable.

Article [10] describes four types of user's fuzzy functions, which are sufficient for entering user's preference of ordinal attributes. It is showed in Figure 1. There are nondecreasing function, nonincreasing function, function in the form of hill, and function in the form of valley. In next text we suppose only these four types of user's fuzzy functions.

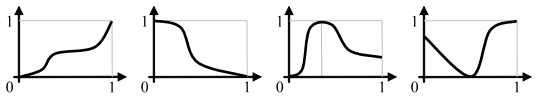


Fig. 1. Four used types of fuzzy functions: nondecreasing, nonincreasing, hill, valley.

Global preferences express how the user U prefers objects from X according to all attributes $A_1, ..., A_m$. On the set of objects X, we consider aggregation function @ with m variables $p_1, ..., p_m$ specified as $@(p_1, ..., p_m) : [0,1]^m \to [0,1]$. For the user U with his user fuzzy functions $f_1^U, ..., f_m^U$, a *user rating function* $@^U$ originates by means of substitution of $p_i = f_i^U(x)$, $i = 1, ..., m$. Then, for every $x \in X$ $@^U(x) = @(f_1^U(x), ..., f_m^U(x))$. With $@^U(x)$ it is possible to evaluate global rating of each object $x \in X$.

With aggregation function, a user U can define the mutual relations of the attributes. For example, we can use *arithmetic average*. For implementation of user's influence to the aggregate function, it is possible to use *weighted average*, where weights $u_1, ..., u_m$ of single attributes $A_1, ..., A_m$ determine how the user prefers single attributes, $@(x) = \frac{u_1 \cdot p_1 + ... + u_m \cdot p_m}{u_1 + ... + u_m}$. When the user does not care about i-th attribute A_i when rating the objects, he can then put 0 into the aggregate function, i.e. $u_i = 0$.

In this work, every user U express his preference as functions $f_1^U, ..., f_m^U$ and $@^U$. Every object $x \in X$ has its own special global rating $@^U(x)$, a measure of pertinence for the user U. In this way, it is possible to find the best K objects for the user U. When there are more objects with the same rating as rating of the best K-th object, a random object is chosen.

3 Usage of B$^+$-tree

For application of local user's preferences we need a data structure, from which it is possible to obtain objects according to user's fuzzy function f^U in descending order according to f^U. Let the objects of the set X be indexed in the B$^+$-tree [2] according to the values of attribute A [1][3][10].

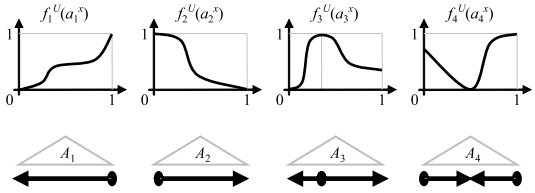


Fig. 2. Ordering interpretation of four used types of fuzzy functions.

Every object $x \in X$ is indexed by the key $k \in dom(A)$, a value of which is equal to a value of attribute a^x of object x, i.e. $k = a^x$. For a key k, in general, more objects with the same A attribute value can exist in X. Therefore, we use the *object array*, in which are stored objects with the same value of the key k. For every key k there is a pointer to corresponding object array.

Since the leaf nodes of the B$^+$-tree are linked in two directions [2], it is possible to cross the B$^+$-tree through the leaf level and to get all the keys. According to course of user's fuzzy function f^U, which is one of four types [10], we can obtain objects in the following different ways (Figure 2).

1. *Nondecreasing function.* In this case, we have to cross the leaf level of the B$^+$-tree from the right to the left. It is possible to get the pairs $(x, f^U(x))$ in the descending order according to the user's preference f^U, because $a^x \leq a^y \Rightarrow f^U(x) \leq f^U(y)$ holds.
2. *Nonincreasing function.* In this case, we have to cross the leaf level of the B$^+$-tree from the left to the right. It is possible to get the pairs $(x, f^U(x))$ in the descending order according to the user's preference f^U, because $a^x \geq a^y \Rightarrow f^U(x) \leq f^U(y)$ holds.
3. *Function in the form of hill.* In this case, we must find key k in leaf level of B$^+$-tree. Key k is the nearest point from maximum of fuzzy function f^U. From the key k we cross the leaf level in two ways concurrently from k to both borders of the B$^+$-tree as in previous two cases.
4. *Function in the form of valley.* In this case, we have to cross the leaf level of the B$^+$-tree in two ways concurrently from both borders to the key k in to inside of leaf level. Key k is the nearest point from minimum of f^U.

In general, f^U cannot be monotone in its domain. In this case, the domain can be divided into continuous intervals (also leaf level of B$^+$-tree), where f^U is monotone on each of these intervals. From the intervals, objects are obtained concurrently according to f^U as well as for monotone fuzzy function [1].

4 Fagin's TA-algorithm

Fagin et al. describe in [4] top-K algorithms, which solve top-K problem without searching of all objects in set X. Algoritms assume that the objects from the set X are, together with values of their m attributes $A_1, ..., A_m$, stored in m lists

$L_1, ..., L_m$. Each i-th list L_i contains pairs (x, a_i^x). Lists $L_1, ..., L_m$ are sorted in descending order according to the values of attributes $a_1^x, ..., a_m^x$ of objects $x \in X$. The aggregate function @ must be monotone according to the ordering in lists [4][1], e.g. weighted average. Then Fagin's algorithms can find the best K objects with regard to existing aggregate function @ without making $|X| \cdot m$ accesses into the lists $L_1, ..., L_m$ [4].

Fagin's algorithm TA (threshold algorithm) needs to access the lists L_1, ..., L_m *sequentially* and also *directly*. With the sequential access, TA searches the lists step by step from the top to the bottom and obtains pairs (x, a_i^x). For every object x, which is detected for the first time in obtained pair (x, a_i^x), TA obtains the missing attribute values of the object x by a direct access to the other lists.

TA uses the list T_K, in which it keeps the best actual K objects ordered according to their value of $@(x)$. During the run of TA a threshold T_h is counted, which is calculated by means of inserting the last seen values of attributes $a_1^{last}, ..., a_m^{last}$ in the lists $L_1, ..., L_m$, respectively, at the sequential access to the aggregate function $T_h = @(a_1^{last}, ..., a_m^{last})$. Rating of the K-th best object in T_K we denote M_K. When $T_h \leq M_K$, TA is able to stop and returns T_K as the list of the best K objects. TA can finish before it comes to the end of all the lists [4]. TA-algorithm is described by the following pseudo-code.

```
Input: Lists L_1, ..., L_m, Aggregation @, int K;
Output: List TK;
var List TK;                    {temporary list of objects}
begin
   while(|TK| < K or T_h > M_K)do
      (x, a_i^x) = getNextPair(L_1, ..., L_m);    {it obtains next pair from
      a_i^last = a_i^x;                            one of the lists L_1, ..., L_m} [1][5]
      T_h = @(a_1^last, ..., a_m^last);
      if(x ∉ TK)then
         get the missing attribute values of the object x;
         if(|TK| < K)then
            insert object x to the list TK on the right place according to @(x);
         else
            if(@(x) > M_K)then
            begin
                delete K-th object from the list TK;
                insert object x to the list TK on the right place by @(x);
            end;
   endwhile;
end.
```

4.1 Application of local preferences

Original Fagin's algorithms offer the possibility of rating objects only globally with an aggregate function @ and to find the best K object for the user U only according to his global preference. For the support of the local preferences, it

is necessary that every i-th list L_i contains pairs $(x, f_i^U(x))$ in descending order according to user's fuzzy function $f_i^U(x)$.

For application of local user's preferences, we use B$^+$-tree (see Section 3). We use as the lists $L_1, ..., L_m$ a list of m B$^+$-trees $B_1, ..., B_m$, where in B$^+$-tree B_i all objects are indexed by values of i-th attribute A_i. Moreover, this structure is common for all users. A particular user U only specifies his local preferences with fuzzy functions $f_1^U, ..., f_m^U$ and global preferences with function $@^U$. Then the algorithm sequentially obtains pairs $(x, f_i^U(x))$ from B$^+$-trees $B_1, ..., B_m$ according to preferences of user U [1].

Algorithm TA also uses the direct access to the lists $L_1, ..., L_m$, where for object x it is needed to obtain its unknown value a_i^x from L_i. B$^+$-tree is not able to make this operation, because it is not possible to obtain the value directly. For a realization of direct access we can use, for example, an associative array.

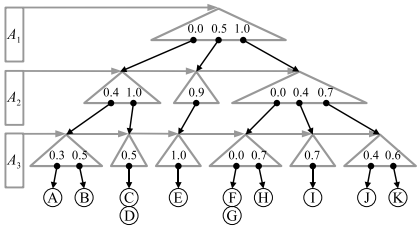
5 Multidimensional B-tree and MD-algorithm

In [1] we describe a top-K algorithm, which solves top-K problem with using the multidimensional B-tree [8] (*MDB-tree*) without searching all the objects. MDB-tree with m levels allows to index set of objects X by attributes A_1, ..., A_m. Each level of MDB-tree is composed from B$^+$-trees containing key values from domain of one attribute. It is showed in Figure 3. Ordering of attributes in levels of MDB-tree is very important [1] (see Section 6).

We use the mark $\rho(k_i)$ for the pointer of the key k in B$^+$-tree in i-th level of MDB-tree. If $i < m$, then $\rho(k_i)$ refers to B$^+$-tree in $(i+1)$-th level of MDB-tree. If $i = m$, i.e. B$^+$-tree is in the last level of MDB-tree, then $\rho(k_i)$ refers to *object array*, where objects with the same values of all the m attributes are stored.

For explicit identification of B$^+$-tree in MDB-tree, we use the sequence of keys, which is called *tree identifier* [1]. An identifier of B$^+$-tree in i-th level is $(k_1, ..., k_{i-1})$. Tree identifier $(\emptyset)$ identifies B$^+$-tree in the first level of MDB-tree.

Furthermore, in MDB-tree we use the *best rating* $B(S)$ of B$^+$-tree S [1]. By the best rating $B(S)$ of B$^+$-tree S with identifier $(k_1, ..., k_{i-1})$ in the i-th level of MDB-tree, we understand a rating of not yet known object x, calculated with $@(a_1^x, ..., a_m^x)$, where the first $i-1$ attribute's values of object x are $k_1, ..., k_{i-1}$ and values of other attributes are 1, i. e. $B(S) = @(k_1, ..., k_{i-1}, 1, ..., 1)$.



	A_1	A_2	A_3
A	0.0	0.4	0.3
B	0.0	0.4	0.5
C	0.0	1.0	0.5
D	0.0	1.0	0.5
E	0.5	0.9	1.0
F	1.0	0.0	0.0
G	1.0	0.0	0.7
H	1.0	0.0	0.7
I	1.0	0.4	0.7
J	1.0	0.7	0.4
K	1.0	0.7	0.6

Fig. 3. Set of eleven objects with values of three attributes stored in MDB-tree.

5.1 MD-algorithm

Obtaining all the objects from MDB-tree is possible with a recursive procedure, which searches MDB-tree in depth-first search. The work [1] includes a sequence of statements with proofs, which shows that it is possible to find best K objects in MDB-tree with the recursive procedure `findTopK` according to a monotone aggregate function @ and without getting all the objects.

Let the keys from the B$^+$-tree S with the identifier $(k_1, ..., k_{i-1})$ be the keys obtained one by one by a run of procedure `findTopK`. The pointer $\rho(p)$ refers to B$^+$-tree P in the next level or to the object array P. Let M_K be the rating of the K-th best object in temporary list T_K of best K objects. If $B(P) \leq M_K$ holds, then the procedure `findTopK(`*MDB-tree*`,` $(k_1, ..., k_{i-1})$`)` can stop in S.

The following pseudo-code describes MD-algorithm.

```
Input: MDBtree MDB-tree, Aggregation @, int K;
Output: List TK;
var List TK;                {temporary list of objects}
begin
   findTopK(MDB-tree, (∅), @, K);  return TK;
end.

procedure findTopK(MDBtree MDB-tree, Identifier (k_1,...,k_{i-1}),
                   Aggregation @, int K);
   while(there is next key in B+-tree)do  {with identifier (k_1,...,k_{i-1})}
      k_i = getNextKey(MDB-tree, (k_1,...,k_{i-1}));
            {ρ(k_i) refers to B+-tree P or it refers to the object array P}
      if(|TK| = K and B(P) ≤ M_K)then return;
      if(P is B+-tree)then
         findTopK(MDB-tree, (k_1,...,k_{i-1},k_i), @, K);
      if( P is object array)then
         while(there is the next object x in P)do
            if(|TK| < K )then
               insert object x to TK to the right place according to @(x);
            else
               if(@(x) > M_K)then begin
                    delete K-th object from the list TK;
                    insert object x to the list TK on the right place by @(x);
               end;
         endwhile;
   endwhile;
end.

procedure getNextKey(MDBtree MDB-tree, Identifier (k_1,...,k_{i-1}));
   choose the next key k_i with next highest value of A_i
   in B+-tree of MDB-tree with identifier (k_1,...,k_{i-1});
   return k_i ;
end.
```

5.2 Application of local preferences

From B$^+$-tree we can obtain keys in descending order according to user's fuzzy function $f_i^U(x)$. Because MDB-tree is composed from B$^+$-trees, it is possible to use application of the local user preferences directly in MD-algorithm by finding the K best objects in MDB-tree. The following procedure `getNextKey` changes the original MD-algorithm.

```
procedure getNextKey(MDBtree MDB-tree, Identifier (k_1,...,k_{i-1}),
                     FuzzyFunction f_i^U);
   choose the next key k_i with next highest value of i-th fuzzy function
   in B+-tree of MDB-tree with identifier (k_1,...,k_{i-1});
   return k_i;
end.
```

6 Combination of TA and MD-algorithm

Here we describe a new top-K algorithm, which is based on combination of MD-algorithm and Fagin's TA-algorithm.

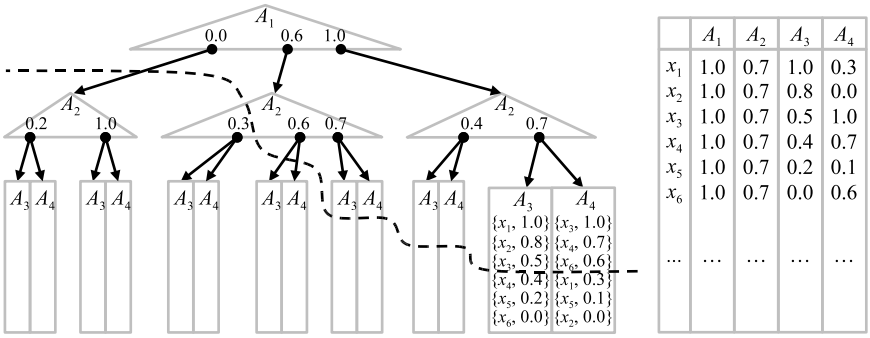
MD-algorithm has the best results, when the objects stored in MDB-tree are distributed regularly (uniform distribution) [1]. When attributes of objects have different size of their actual domains, order of attributes in levels of MDB-tree is very important for efficiency of MD-algorithm. It is better for MD-algorithm to build MDB-tree with smaller actual domains in its higher levels and attributes with bigger actual domains in its lower levels. When most of the object's attributes have their actual domains of big sizes, the usage of MD-algorithm is not suitable solution of top-K problem. In this case, the usage of TA-algorithm is more suitable. In general, we can suppose according to size of attribute's domains of a known object set, which of the algorithms will achieve better results.

Example 1. We have set of 8 822 flats for rent in Prague. Flats have four important attributes for users, *District*, *Type*, *Area*, and *Price*. These attributes have sizes of domains, $\|dom(District)\| = 10$, $\|dom(Type)\| = 10$, $\|dom(Area)\| = 229$, $\|dom(Price)\| = 411$, respectively. When a user prefers attributes *District* and *Type*, then it is better to store flats in MDB-tree and to use MD-algorithm. On the other hand, when a user prefers attributes *Area* and *Price*, then it is better to use TA-algorithm and to store flats in Fagin's lists.

In general, the attribute with a small domain size is *nominal attribute* and attribute with big domain size is *ordinal attribute* (see Section 2). It is valid also in Example 1, attributes *District* and *Type* are nominal attributes, *Area* and *Price* are ordinal attributes.

6.1 MXT-algorithm

Therefore we developed a new top-K algorithm, *MXT-algorithm*, which is based on combination of MD-algorithm and Fagin's TA-algorithm. MXT-algorithm



	A_1	A_2	A_3	A_4
x_1	1.0	0.7	1.0	0.3
x_2	1.0	0.7	0.8	0.0
x_3	1.0	0.7	0.5	1.0
x_4	1.0	0.7	0.4	0.7
x_5	1.0	0.7	0.2	0.1
x_6	1.0	0.7	0.0	0.6
...	...	...	...	...

Fig. 4. Two nominal attributes are stored as MDB-tree and two ordinal attributes are stored as Fagin's lists. Under dotted line there is part of the data structure, in which the MXT-algorithm does not access during its computation.

uses a new data structure, which is composed of MDB-tree and Fagin's sorted lists. This data structure is shown in Figure 4.

We suppose a set of objects X with m attributes $A_1, ..., A_m$. Attributes $A_1, ..., A_n$ are nominal attributes and $A_{n+1}, ..., A_m$ are ordinal attributes. Attributes $A_1, ..., A_n$ are stored in MDB-tree with n levels. Instead of the following $m - n$ levels of MDB-tree, groups of $m - n$ Fagin's sorted lists are used. These lists contain pairs (x, a_i^x) with values of attributes $A_{n+1}, ..., A_m$.

MXT-algorithm is developed on the base of MD-algorithm. First n attributes $A_1, ..., A_n$ are searched in the same way as during the computation of MD-algorithm. In every B^+-tree there are references into the groups of $m-n$ Fagin's sorted lists. In each of these groups a new instance of TA-algorithm is run.

The efficiency of MXT-algorithm is based on idea that we do not need to obtain the best K objects from each the group of $m - n$ Fagin's sorted lists. We need to obtain only objects with better rating as M_K (rating of the K-th best object in global list T_K). There, it is sufficient that local threshold T_h^{local} (in each the group of $m - n$ Fagin's sorted lists) with M_K is compared. Local TA-algorithm is able to stop earlier in group of Fagin's sorted lists, $T_h^{local} \leq M_K$ holds earlier than in original TA-algorithm. (see Figure 4, dotted line).

The following pseudo-code describes the MXT-algorithm, where procedure `getNextKey` is the same as in the MD-algorithm (see Section 5).

```
Input: MDBtree MDB-tree, Aggregation @, int K;
Output: List TK;
var List TK;                {temporary list of objects}
begin
   findTopK(MDB-tree, (∅), @, K);
   return TK;
end.
```

```
procedure findTopK(MDBtree MDB-tree, Identifier (k_1, ..., k_{i-1}),
                   Aggregation @, int K);
   while(there is next key in B+-tree)do        {with identifier (k_1, ..., k_{i-1})}
      k_i = getNextKey(MDB-tree, (k_1, ..., k_{i-1}));
            {ρ(k_i) refers to B+-tree P or it to group of Fagin's lists P}
      if(|TK| = K and B(P) ≤ M_K)then return;
      if(P is B+-tree)then
         findTopK(MDB-tree, (k_1, ..., k_{i-1}, k_i), @, K);
      if( P is group of Fagin's lists)then
         while(|TK| < K or T_h^local > M_K)do
            (x, a_i^x) = getNextPair(L_1, ..., L_m); {it obtains next pair from
            a_i^last = a_i^x;                         one of the lists L_1, ..., L_m}
            T_h^local = @(a_1^last, ..., a_m^last);
            if(x ∉ TK)then
               get the missing attribute values of the object x;
               if(|TK| < K)then
                  insert object x to the list TK on the right place by @(x);
               else
                  if(@(x) > M_K)then
                     begin
                        delete K-th object from the list TK;
                        insert object x to the list TK oaccording to @(x);
                     end;
         endwhile;
   endwhile;
end.
```

6.2 Application of local preferences

Analogously to MD-algorithm in attributes $A_1, ..., A_n$ it is possible to use application of the local user preferences. Procedure `getNextKey` changes MXT-algorithm in the same way as in the MD-algorithm (see Section 5.1).

In attributes $A_{n+1}, ..., A_m$ it is also possible to use application of the local user preferences. Analogously to TA-algorithm, we use as the group of $m-n$ Fagin's lists $L_{n+1}, ..., L_m$, a list of $m-n$ B$^+$-trees $B_{n+1}, ..., B_m$, and for a realization of direct access we can use, for example, an associative array.

7 Implementation and Experiments

We implemented top-K TA-algorithm, MD-algorithm and MXT-algorithm using lists based on B$^+$-trees. The implementation has been developed in Java and it uses data structures created in memory. Important for us was the number of accesses into used data structures during calculation of top-K algorithms.

We tested TA-, MD- and MXT-algorithm. During the tests, we used uniform preferences. We used user's fuzzy functions with course "$f(x)$=x" as user's local preferences, and we used the arithmetic average as user's global preference.

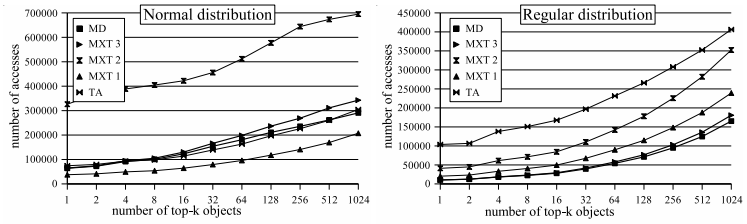


Fig. 5. Normal and regular distribution of the attributes values.

At first, we tested two sets of 100 000 objects with 5 attributes with normal and regular (uniform) distribution of attribute values. We used TA-algorithm, MD-algorithm and three variants of MXT-algorithm, i.e. MXT 3, MXT 2, MXT 1, because attribute types were not important there. For example, variant MXT 3 uses first 3 attributes as nominal attributes, which are stored as MDB-tree with 3 levels and other 2 attributes are stored as groups of 2 Fagin's sorted lists. Figure 5 shows results of this test.

The best results have been achieved with MXT 3 and MD-algorithm for the set of objects with the regular distribution of the attributes values. Test for sets of objects with normal distribution of attribute values has shown that the new MXT-algorithm can in some cases also achieve worst results.

Afterwards, we tested the sets of 8 822 flats for rent in Prague (see Section 6, Example 1). There were two nominal attributes with a small domain size and two ordinal attributes with a big domain size. We used TA-algorithm, MD-algorithm and the most suitable variant of MXT-algorithm (as MXT 2 in previous test). Figure 6 shows results of this test for uniform preferences, and for real user's preferences, where user prefers flats of some types in specific districts, with smaller prices and bigger areas. These two ordinal attributes were preferred according to non-monotone fuzzy functions.

The best results has been achieved with MXT-algorithm for both of used preferences. The test has shown that MXT-algorithm is most efficient solution of top-K problem for some cases. Especially it is efficient for a set of objects, which has nominal attributes with a small domain size and several ordinal attributes with a big domain size.

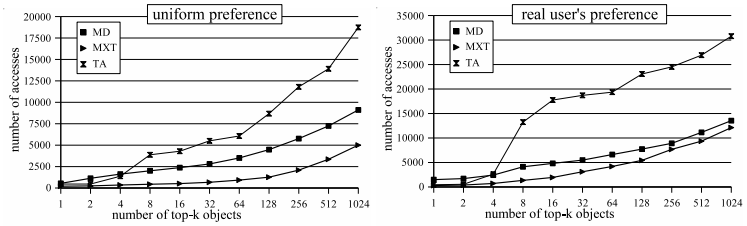


Fig. 6. Number of accesses objects during finding the best flats in Prague.

8 Conclusion

We developed a new MXT-algorithm, which can effectively find the best K objects by user's preferences without accessing all the objects. We implemented top-K algorithms TA-, MD- and MXT-algorithm with support of user's preferences. Results of MXT-algorithm have shown to be comparable with those obtained by others used top-K algorithms.

MXT-algorithm is based on combination TA- and MD-algorithm. According to type of object attributes it is possible to store a set of objects in MDB-tree, in Fagin's lists or in data structure, which MXT-algorithm uses. Moreover, MXT-algorithm can find the best K objects in each of these data structures. In this meaning, TA- and MD-algorithm are extreme cases of MXT-algorithm.

Motivation of future research can be to find application of developed algorithms in various data structures. The article [11] deals with solving of the top-K problem in XML. Application of our algorithms in XML might be also interesting.

References

1. Ondreička, M., Pokorný J.: Extending Fagin's algorithm for more users based on multidimensional B-tree. In: Proc. of ADBIS 2008, P. Atzeni, A. Caplinskas, and H. Jaakkola (Eds.), LNCS 5207, Springer-Verlag Berlin Heidelberg, 2008, pp. 199214.
2. Bayer, R., McCreight, E.: Organization and Maintenance of Large Ordered Indices, Acta Informatica, Vol. 1, Fasc. 3, 1972 pp. 173-189.
3. Eckhardt, A., Pokorný, J., Vojtáš, P.: A system recommending top-k objects for multiple users preference. In Proc. of 2007 IEEE International Conference on Fuzzy Systems, July 24-26, 2007, London, England, pp. 1101-1106.
4. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. Journal of Computer and System Sciences 66 (2003), pp. 614-656.
5. Gurský, P., Lencses, R., Vojtáš, P.: Algorithms for user dependent integration of ranked distributed information. Proceedings of TED Conference on e-Government (TCGOV 2005), Pages 123-130, 2005.
6. Bruno, N., L. Gravano, L., Marian, A.: Evaluating top-k queries over web-accessible databases. in: Proc. of International Conference on Data Engineering (ICDE), 2002, pp. 369 - 380.
7. Chaudhuri, S., Gravano, L., Marian, M.: Optimizing Top-k Selection Queries over Multimedia Repositories. IEEE Trans. On Knowledge and Data Engineering, August 2004 (Vol. 16, No. 8) pp. 992-1009.
8. Scheuerman, P., Ouksel, M.: Multidimensional B-trees for associative searching in database systems. Information systems, Vol. 34, No. 2, 1982.
9. Vojtáš, P.: Fuzzy logic aggregation for semantic web search for the best (top-k) answer, Capturing Intelligence, Volume 1, Chapter 17, 2006, Pages 341-359.
10. Gurský P., Vaneková V., Pribolová J.: Fuzzy User Preference Model for Top-k Search. Proceedings of IEEE World Congress on Computational Intelligence (WCCI), Hong Kong, FS0377, 2008.
11. Marian, A., Amer-Yahia, S., Koudas, N.: Adaptive Processing of Top-k Queries in XML. Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on 05-08 April 2005 Page(s):162 - 173.