Speeding up the NRA algorithm

Peter Gurský[1] and Peter Vojtáš[2]

[1] University of P.J.Šafárik, Košice, Slovakia
[2] Charles University, Prague, Czech Republic
peter.gursky@upjs.sk,peter.vojtas@mff.cuni.cz

Abstract. Methods of top-k search with no random access can be used to find k best objects using sorted access to the sources of attribute values. In this paper we present new heuristics over the *NRA* algorithm that can be used for fast search of top-k objects using wide range of user preferences. *NRA* algorithm usually needs a periodical scan of a large number of candidates during the computation. In this paper we propose methods of no random access top-k search that optimize the candidate list maintenance during the computation to speed up the search. The proposed methods are compared to a table scan method typically used in databases. We present results of experiments showing speed improvement depending on number of object attributes expressed in a user preferences or selectivity of user preferences.

Key words: top-k search, no random access, *TA*-sorted variants, user preferences

1 Introduction

Top-k search became popular with growing use of the web services and increasing datasets sizes. Users are usually interested in few best objects rather than a big list of objects as a result. Top-k algorithms usually follow two main goals. Firstly, they minimize the number of source data to be processed i.e. they find the correct top-k objects using only a part of data. Secondly, the algorithms tend to minimize the number of accesses to the sources and the computation time as well.

1.1 Motivation

In the family of Threshold algorithms (*TA*) the basic assumption is a monotone combination function over ordered sources. The expressivity of a monotone combination function over domain ordering is low. Many authors [22, 23] face the problem of more expressive queries by complicate analysis of ranking functions and they offer a kind of multidimensional search.

Example 1. Imagine a user u_1 looking for a low price flat with size about $60m^2$. His overall ranking function F can be expressed as follows:

$$F_{u_1}(price, size) = (1 - \frac{price}{max_{price}}) \cdot max\{0, (1 - |1 - \frac{size}{60}|)\} \qquad (1)$$

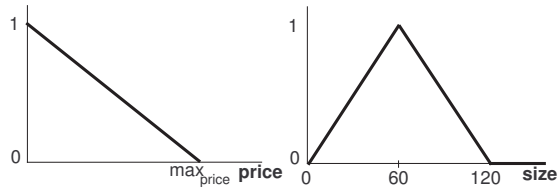

Fig. 1. Local preferences of user u_1

We can see that F_{u_1} is not a monotone function and *TA* cannot to be used ad-hoc.

Instead of a difficult function analysis we prefer a different form of query compound of local preferences and a monotone combination function. Local preference represents user's notion about the suitability of values from the attribute domain. Local preference can be expressed e.g. by fuzzy predicate that maps attribute domain into the interval $[0, 1]$, where 1 means the most preferred domain value and 0 means the least preferable value.

The monotone combination function is used to compare objects incomparable in particular local preferences (one flat can be better in price, another one in size). Typical monotone combination functions are weighted average, minimum, maximum or a set of ranking rules [10].

The ranking function F_{u_1} of user u_1 can be written as a monotone combination (product) of partially linear scoring functions f_p and f_s as depicted on Figure 1.

Formally the preferences of user u for a set of objects with m attributes $a_1, \ldots, a_m$ are described by m arbitrary scoring functions of one variable $f_{a_1}, \ldots, f_{a_m}$ (local preferences) and one monotone combination function F (global preferences). For every object X with attribute values $x_1, \ldots, x_m$ the preference of user u for object X is equal to $F(f_{a_1}(x_1), \ldots, f_{a_m}(x_m))$.

In the naïve approach, we could sort both attributes according to local preferences and use any effective *TA*-like algorithm to find top-k flats. Unfortunately, the meaning of good values of attributes (specified by local preferences), as well as the combination function, can be different for each user.

Example 2. Consider a user u_2 with preferences for rather large flats but mainly the flats with the price about \$50k:

$$\begin{aligned} &F_{u_2}(price, size) = 3 \cdot f_{p_2}(price) + f_{s_2}(size), \text{where} \\ &f_{p_2}(price) = \begin{cases} \frac{price}{10k} - 4 & , price \in \langle 40k, 50k \rangle \\ 6 - \frac{price}{10k} & , price \in (50k, 60k) \\ 0 & , \text{otherwise} \end{cases} \\ &f_{s_2}(size) = \frac{size}{max_{size}} \end{aligned} \tag{2}$$

We can see that the local preferences induce different ordering of the attributes than functions on figure 1. Now the naïve approach fails - reordering in the time of the query is unacceptable.

In this paper we present the heuristics for *3P-NRA* algorithm, which is similar to *NRA* algorithm - a member of the Threshold algorithms. We use the *NRA*-like algorithms because they work with data obtained by sorted access from the sources ordered from the best to the worst in particular attributes and as presented in [10] sorted access can be simulated using an index structure depending on an attribute type. For an ordinal attribute a B+ tree traversed according to user local preference can be used. In example 2 the price tree is traversed from maximal value in descending order using leaf pointers. In the size tree we have to create two pointers, both starting at $60m^2$. First pointer traverses the tree in descending direction and the second one in ascending direction. For a simulation we need to identify local maxima, then the scoring function is used as a black box. Having simple functions (partially linear) we can easily find points of extremes to identify the pointers and directions. Arbitrary functions expressing local preferences would need function analysis. This analysis is much more simple than in case of multidimensional analysis used in [22]. Besides ordinal and nominal attributes presented in [10] we can simulate sorted access over metric and hierarchical attributes, too (not published yet).All simulation algorithms have their speed close to a simple scan of tables sorted in.

In order to enable different users to express top-k queries we designed an intuitive user friendly interface. As an alternative to complicated input users can also use an inductive procedure. Our system was integrated in the *UPRE* system [9]. In *UPRE*, user can define his/her preference by ordering or evaluating a sample of objects. The system creates input for top-k search based on user's evaluation in a two step learning process. First, local fuzzy preference functions have to be induced for each attribute. Second, using these functions, the monotone combination function in the form of fuzzy rules is learned by an inductive logic programming method *IGAP* described in [12]. Alternatively we can use the SVM based system [21].

1.2 Contribution

Having simulations of sorted access it is natural to find top-k objects using the *NRA* algorithm requiring sorted access and monotone combination function only. The original *NRA* algorithm is rather slow because of inefficient management of candidates during the computation. In this paper we present several variants of *NRA* algorithm that speed up the computation of top-k objects.

- Our first contribution includes new heuristics in *3P-NRA* algorithm (an extension of *NRA* algorithm) named 3P-NRAz and 3P-NRA2z that make the computation fast and more stable than the previous approaches.
- The second contribution is the execution of experiments in different directions. We compared time efficiency of a table scan over joined data to several

variants of our *3P-NRA* algorithm with data in one index per attribute architecture. We describe common situations in which our approach is better than simple table scan. In section 3 we also show that the *3P-NRA* algorithm with heuristics outperforms the previous *NRA* algorithm significantly.

2 3P-NRA algorithm

In this section we present the modification of *NRA* algorithm [6] named *3P-NRA* (3-phased no random access) proved to be better than *NRA* in several aspects. The presentation of *3P-NRA* algorithm without the support of local preferences is accepted for ADBIS conference [8].

In the rest of this paper we use the following notation. Value m represents the number of attributes of objects or the number of sources. An arbitrary object is denoted as $X = (x_1, \ldots, x_m)$, where $x_1, \ldots, x_m$ are real attribute values of X. $f_1, \ldots, f_m$ are arbitrary scoring functions of one variable that represent local preferences to attribute values. For each $i \in \{1, \ldots, m\}$, $f_i(x_i)$ represents a user preference to the real value of the i-th attribute of X. Let $V(X) = \{i_1, \ldots, i_n\} \in \{1, \ldots, m\}$ be a subset of known attributes $x_{i_1}, \ldots, x_{i_n}$ of X, we define $W_V(X)$ (or shortly $W(X)$ if V is known from context) to be minimal (worst) possible value of the combination function F for the object X. We assume that F is a monotone combination function. We compute $W_V(X)$ so that we substitute each missing attribute by the minimum of appropriate scoring function. For example if $V(X) = \{1, \ldots, g\}$ then $W_V(X) = F(f_1(x_1), \ldots, f_g(x_g), min(f_{g+1}), \ldots, min(f_m))$.

Analogously we define maximal (best) possible value of the combination function F for object X as $B_V(X)$ (or shortly $B(X)$ if V is known from context). Since we know that sorted access returns values in descending order, we can substitute each missing value by the corresponding value from the vector $u = (u_1, \ldots, u_m)$, where $u_1, \ldots, u_m$ are the last scoring values seen from each source. For example if $V(X) = 1, \ldots, g$ then $B_V(X) = F(f_1(x_1), \ldots, f_g(x_g), u_{g+1}, \ldots, u_m)$.

The real value of the object X is $W(X) \leq F(f_1(x_1), \ldots, f_m(x_m)) \leq B(X)$. Note that during the computation unseen objects (no values are known) have $B(X) = F(u_1, \ldots, u_m)$. The value $\tau = F(u_1, \ldots, u_m)$ is known as the threshold value.

In the algorithm we use the top-k list T ordered by the worst value. The object in T with the smallest worst value is labeled T_k. In the (unordered) set C we store the candidates with the worst value smaller or equal to $W(T_k)$ but with the best value larger than $W(T_k)$. These are the objects with a chance to get into T later. We call the objects in C candidates.

In *3P-NRA* we implemented C as a hash table with object identifiers as a key. To recognize the sources with all known values among the objects in $T \cup C$ we maintain the number of missing values for each source.

3P-NRA algorithm works as follows:

Input: k, F, m sources
Output: k ranked objects if exist

$T = \emptyset$, $C = \emptyset$
Phase 1:
Do the sorted access in parallel to all sources.
For every object X seen under sorted access compute $W(X)$ and do
    If $|T| < k$ then put X to T
    Else If $W(X) > W(T_k)$ then
            If $X \notin T$ move T_k to C, put X to T
        Else put X to C
If $W(T_k) \geq$ threshold τ goto Phase 2
repeat Phase 1
Phase 2:
Do the sorted access in parallel to the sources for which there are unknown values for objects in C and T.
For every object X seen under sorted access do
    If $X \notin T \cup C$ ignore it
    Else If $B(X) \leq W(T_k)$ remove X from C
        If $|C| = 0$ return T and exit
        If $W(X) > W(T_k)$ and $X \notin T$ move T_k to C, put X to T
If ($W(T_k)$ increased) OR (threshold τ decreased) then
   heuristic H can choose to go to Phase 3
repeat Phase 2
Phase 3:
For every object $X \in C$ compute $B(X)$;
If X is no more relevant (i.e. $B(X) \leq W(T_k)$) remove X from C
If $|C| = 0$ return T and exit; otherwise goto Phase 2.

Phase 1 works similarly to the standard *NRA* algorithm with the exception of the threshold test and the absence of candidate set search. The heuristic H in *3P-NRA* algorithm can be used to skip an expensive computation of phase 3. On the other hand, if H always chooses to do the phase 3 the *3P-NRA* algorithm is proved to be instance optimal. The instance optimality of *NRA* means, that if *NRA* finds top k objects using y sorted accesses, then there are no algorithms reading the sources only by sorted access, that can find top k objects using less than $\frac{y}{m}$ sorted accesses (see [6]).

Theorem 1. *Let F to be a monotone combination function, then algorithm* 3P-NRA *correctly finds top-k objects.*

Theorem 2. *Let F be a monotone combination function. If heuristic H always chooses to go to phase 3, then algorithm* 3P-NRA *makes at most the same number of sorted accesses as* NRA *algorithm i.e.* 3P-NRA *algorithm is instance optimal.*

Proofs excluding the local preferences of can be found in [8]. The improvements of the *3P-NRA* algorithm in contrast to *NRA* [6] are the following:

- New objects are considered in phase 1 only. Other objects are ignored. This is the most significant difference between *3P-NRA* and *NRA*.
- Many computations of the best values are omitted. This feature allows the speedup of the computation especially for large number of candidates.
- After acquisition of all unknown values of any attribute among objects in $T \cup C$ the algorithm stops working with the corresponding source (no more sorted accesses to the source will be done). This feature decreases the number of disk accesses significantly.
- A good choice of heuristic H can yield a massive speedup of the algorithm, however it can slightly increase the number of disk accesses according to H.

In [8] we discussed also the modification of 3P-NRA algorithm for the cases when the candidate set C do not fit into the available memory. The proposed modification allows efficient computation of top-k objects with less than 1/100 of candidates in memory.

2.1 Heuristics

Two the most expensive points of instance optimal *3P-NRA* algorithm are partial join of data form the sources and the computation of the best values in the candidate set C. All heuristics presented in this section follow the idea of decreasing the number of expensive best values computation in phase 3. Nevertheless, we need to do this computation occasionally to prevent the processing of all input data.

The first approach to decrease the number of computations of the best values in phase 3 is to use the heuristic H mentioned in phase 2. In our tests in section 3 we use the heuristic that goes to phase 3 only each 1000^{th} loop of phase 2. A number 1000 is an approximated number of sorted accesses to the sources that is done by reading one disk page per attribute when user preferences cover 5 attributes of the objects. Thus we can be near the optimal number of disk accesses. In the tests we show orders of magnitude speedup against algorithms without the heuristic. This heuristic was used in [8] too. We will call the algorithm having this heuristic *3P-NRA2*.

Our new heuristic of decreasing the number of computations of the best values modifies the phase 3 as follows:

Phase 3:

```
Iterate over objects X ∈ C and compute B(X)
  If X is relevant (i.e. B(X) > W(T_k))
    break the cycle and goto Phase 2
  Else Remove X from C
       numberOfRemoved++
  If |C| = 0 return T and exit;
  If numberOfRemoved ≥ 100
    Create new set CNew
    For every object X ∈ C compute B(X)
      If X is relevant, copy X to CNew
```

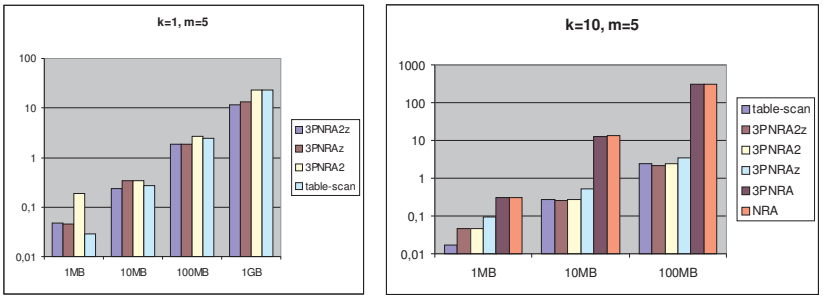

Fig. 2. Computation time in seconds over different size of datasets

```
If |CNew| = 0 return T and exit;
Free memory used by C, set C = CNew
Set numberOfRemoved = 0 and goto Phase 2;
```

The main idea behind this heuristic is to skip the computation of the best values when there are still many relevant candidates. By iteration through random elements of set C there is a high probability that after erasing of 100 not relevant objects the candidate set C will have less than 1% of relevant objects. It is more effective to create a new small set rather than deletion of 99% objects from C. We will call this algorithm *3P-NRAz*. Note that the algorithm having this modification of phase 3 is correct and instance optimal. Both the correctness and the instance optimality come from the fact that every time the previous version of phase 3 returns list T, so does the new version.

We have also the possibility of combination of proposed heuristics. The result of the combination is the algorithm called *3P-NRA2z*. This new algorithm is the best in our tests especially for bigger data sizes and restrictive local preferences.

3 Experiments

This section reports the experiments with algorithms using only sorted access over artificial data. The experiments were conducted on a PC having Intel Core 2 1,86 GHz CPU and 2 GB RAM running on Windows XP. We minimized the number of other processes during the tests to keep a stable test environment. Algorithms are implemented in Java. The size of the disk pages was set to 4 kB. All table scan computations were tested using InnoDB table organized as a heap file with no index support in MySQL 6 database.

In the first experiment we compare the mentioned algorithms over different size of datasets: 1 MB (10k objects), 10 MB (100k objects), 100 MB (1M objects) and 1GB (10M objects). Each dataset consists of 5 attributes $x_1, \ldots, x_5$. Each attribute has values in range [0, 1] and Gaussian distribution (the most difficult distribution for *TA*-like algorithms). Tests were computed with 5 different com-

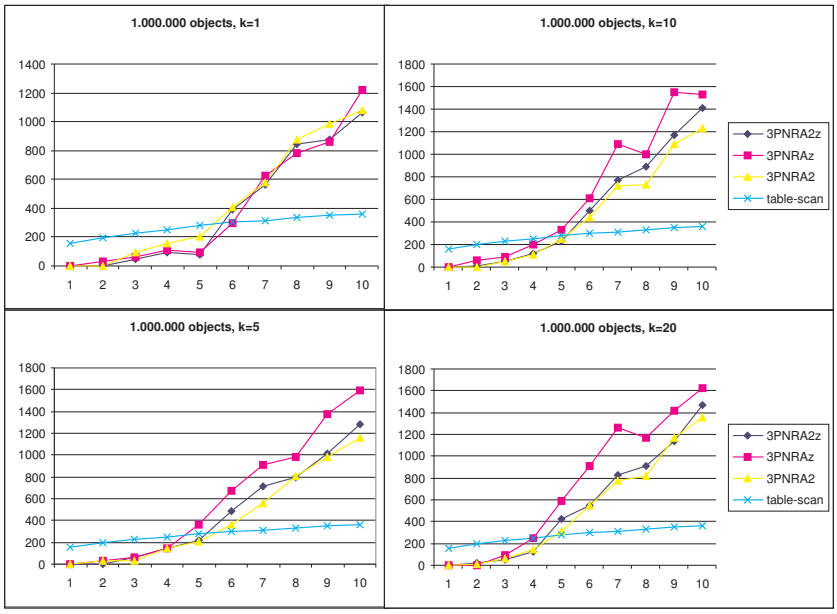

Fig. 3. Computation time in ms over different number of attributes in user preferences

bination functions $F(X) = \sum_{i=1}^{5} a_i \cdot x_i$ where a_i were randomly generated from interval $\langle 1, 5 \rangle$. Results presented on figure 2 are average times of all the tests. *NRA* and *3P-NRA* algorithms without any heuristic are significantly worse than the algorithms with heuristics. For example the computation of top-10 objects over 100MB dataset with *NRA* algorithm took 308 seconds. The same result was computed in 2 seconds using the fastest *3P-NRA2z* algorithm. Table scan took 2,5 seconds. Because of the low performance of the original *NRA* algorithm and instance optimal *3P-NRA* algorithm we do not compare them in all our tests. The graph on the left shows the computation time needed to search the best object from the datasets. The proposed methods become more effective in comparison to table scan with the growing data size. For example the table scan over 1GB dataset takes more than 23 seconds while the *3P-NRA2z* algorithm needs 11,5 seconds. We have to admit that with the growing k the computation times become longer than table scan times e.g. when $k = 20$ the *3P-NRA2z* algorithm needs 42 seconds to present the results, whilst the table scan time remains 23 seconds.

Our second experiment was motivated by our experiences in project NAZOU (see `http://nazou.fiit.stuba.sk`). In NAZOU, our target domain are job offers. We identified more than 15 attributes of job offers. Users usually express their preferences by a small subset of attributes (2 to 5). In this experiment we used 10 randomly generated attributes with uniform distribution of values having 1 million tuples (100MB). The base combination function used in this

experiment was $F(X) = 3x_1+2x_2+x_3+2x_4+2x_5+3x_6+2x_7+x_8+2x_9+2x_{10}$. In each run of the test we have a different number of the attributes. The test with m attributes uses the first m elements of the combination function. From the results presented on Figure 3 we can conclude that our methods of top-k search become efficient when user preferences covers up to 5 attributes. These advantages of top-k search grow with the number of stored attributes. For example cars, notebooks or mobile phones are domains with more than 25 attributes.

In NAZOU we also face the problem of one object having more than one value of an attribute e.g. required education level of a job offer can be expressed as "bachelor or master degree". When computing the overall value of an object we use the fuzzy value of more preferred attribute value. For example in this attribute the mentioned job offer would suit a young man who has just ended the bachelor studies. In this experiment each source has 2 values of attribute per object. By joining the data from 5 sources holding 50k objects (100k values per source), we got 1.6M tuples. Results in table 1 show the significant difference between table scan of joined data and our methods. Even low effective algorithms *NRA* and *3P-NRA* are faster than table scan when searching the best object only.

Table 1. Computation time in ms over data with 2 attribute values per object over 100.000 objects.

k	1	5	10	20
3PNRA2z	157	203	218	250
3PNRAz	234	297	422	484
3PNRA2	141	188	203	235
3PNRA	3782	4484	9031	10890
NRA	3766	4500	9078	10875
table-scan	5890	5890	5890	5890

Another significant property that influences the computation time of top-k search is the selectivity of user local preferences i.e. the number of values having minimal fuzzy value equal to zero. The situation with many zeros is typical also when the attribute values are unknown. In this case they can be marked as not preferred values (we need some value to compute the overall value of the objects). Unknown values are typical for web extracted data.

Domain values having minimal fuzzy value do not need to be processed from the sources. After reaching the first object with the minimal fuzzy value, the unknown values of this attribute can be substituted by the minimal fuzzy value in all objects not seen in the given attribute. In Figure 4 there are the results of tests with 5 attributes over sources with uniform distribution. The combination function in this experiment was $F(X) = 3f_1(x_1) + 2f_2(x_2) + f_3(x_3) + 2f_4(x_4) + 2f_5(x_5)$ where $f_i = max\{0, \frac{100 \cdot x_i - d}{100 - d}\}$, and d have values $0.05, \ldots, 0.95$. It is not surprising that for high selective local preferences the top-k search is better

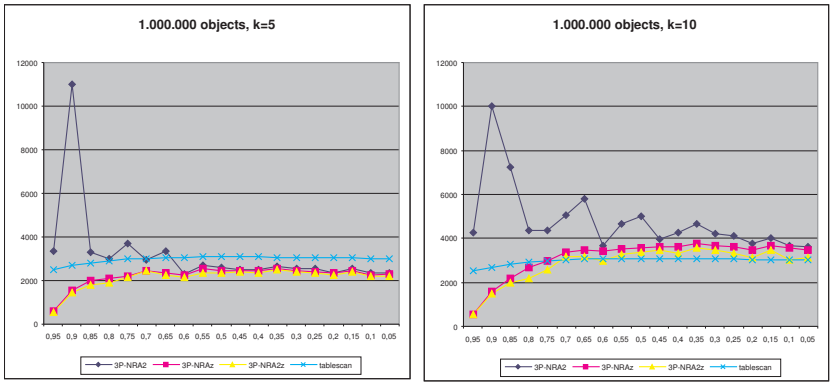

Fig. 4. Computation time in ms over different selectivity of local preferences. X-axis represents the number of objects that have fuzzy value zero (in %)

than table scan. Results show an interesting behavior of *3P-NRA2* algorithm for high selective queries. We have analyzed it and found out that after a short phase 1 there was a long and expensive computation of phases 2 and 3 with the candidate set of high cardinality. Even if the number of computations of phase 3 has decreased 1000 times compared to *NRA* algorithm, the number of best value evaluations is still high. The new heuristic with restrictive phase 3 is very stable and resistant against long-term large candidate set.

All tests indicate the situations in which the *3P-NRA* style of computation is suitable. All the mentioned situations are typical for the advanced web based search applications that allow complicate user preferences. Users are usually interested in a few best objects only and express their preferences with a small part of attributes. Local preferences are often restrictive (they prefer specific job positions or they want to find houses in a small region). Data sets of objects grow every day. Objects can have more values of one attribute and many attribute values are unknown.

The original *TA*-like algorithms were developed to aggregate the data from the distributed sources having the form of web services. If we combine all the features of data and queries, the use of *3P-NRA* algorithm with heuristics becomes profitable even for local repositories.

4 Related work

In our approach we face the problem of complex queries by simulation of sorted access over various attribute types. The simulation is directed by an arbitrary local preference function. The combination of arbitrary local preference functions and monotone combination function has a high expressive power.

Top-k query processing using monotone combination function and ordered sources or sources accessed only by random access was extensively studied [1–

3, 6, 11]. If we have ordered sources, many of these algorithms are proven to be instance optimal for the top-k search of a single user (constant local preferences).

The top-k algorithms embedded in RDBMS [14, 15] are concerned with augmenting the query optimizer to considering rank-joins during plan evaluation. The rank-join algorithms require ordered data on input similarly to the previous middleware algorithms.

The problem of more expressive queries was studied in different ways. Systems MPro [5] and Upper [4] access unordered sources only by random access. In system PREFER [13] the sorted access is provided by choosing one of several prepared ranked materialized views having ordering near to the ordering made by ranked query.

Zhang et al in [23] present the OPT* algorithm combining discrete selection condition and continuous optimization over arbitrary ranking function to find the first best object. Xin et al [22] analyze the ranking function of many variables similarly to [23] to navigate through the huge set of states over m B+ trees. If we can analyze the ranking function over any domain subregion (to find the minimum and possibly recognize monotonicity) this approach should be able to find top-k objects in an effective way. Authors in [22] made many tests and compared their approaches to the table scan as we did. Nevertheless it is difficult to compare our results with theirs because of questionable results of their tests. For example the table scan over 1M tuples having 3 attributes took over 30 seconds in their tests. Our results of the table scan over 1M tuples with 5 attributes shows times from 2 to 3 seconds on a common workstation machine.

Another possible view of recent research distinguishes two branches. One branch of research generalizes types of data to uncertain (see e.g. [19, 18]) or to XML data (see e.g. [20]). The other branch focuses on top-k query optimization.

5 Discussion

In this paper we have studied methods of top-k search with no random access which can be used to find k best objects using sources that can be read only by sorted access. Such methods usually need to work with a huge set of candidates during the computation and management of candidates is an important issue.

We have introduced new methods for top-k search with sorted access and implementation of own experimental environment. New versions of *3P-NRA* algorithm were tested with different heuristics that improve the computational time.

We can say that it is suitable to use our methods of top-k search for systems allowing various user preferences. Our motivation comes from querying a big repository with data extracted and/or retrieved from the web relevant to job search. Such data typically contain several dozens of different attributes and single user query typically uses a constant fraction of them, nevertheless all of them are used by some user. System shows better performance if objects have more values of the same attribute. Top-k search with the proposed algorithms should be preferred when the specified local preferences are restrictive.

Moreover the combination of the methods presented in this paper together with the sorted access simulation algorithms over ordinal, nominal, hierarchical and metric attributes becomes a powerful tool for top-k search with a high expressive power [10].

Our system uses a light weight proprietary disk data structures used to simulate the sorted access. This serves only to top-k search, we do not need the full functionality of a transactional multiuser RDBMS.

In the project NAZOU (see `http://nazou.fiit.stuba.sk`), we use the sorted access to process the output from different methods, which produce data with respect to user preferences to attribute values. There are different methods over various index structures for different types of attributes: ordinal, nominal, hierarchical and metric attributes. To obtain the combination function for different users, our system uses learning user preferences from a sample of objects (see [12]) or a recommendation for a similar user based on it. The learned preferences have a form of monotone fuzzy rules (instead of the analytical form).

Considering that users are interested only in a part of the attributes and that attribute domains can be complex structures, the new methods of top-k search become very useful.

References

1. Akbarinia, R., Pacitti, E., Valduriez, P.: Best Position Algorithms for Top-k Queries. In VLDB (2007)
2. Bast, H., Majumdar, D., Schenkel, R., Theobald, M., Weikum, G.: IO-Top-k: Index-Access Optimized Top-k Query Processing. In VLDB (2006)
3. Balke, W., Güntzer, U.: Multi-objective Query Processing for Database Systems. In VLDB (2004)
4. Bruno, N., Gravano, L., Marian, A.: Evaluating top-k queries over web-accessible databases. In ICDE (2002)
5. Chang, K.C.C., Hwang S.W.: Minimal probing: Supporting expensive predicates for top-k queries. In SIGMOD (2002).
6. Fagin, R., Lotem, A., Naor, M.: Optimal Aggregation Algorithms for Middleware. In ACM PODS (2001)
7. Gurský, P., Lencses, R., Vojtáš, P.: Algorithms for user dependent integration of ranked distributed information. In TCGOV (2005)
8. Gurský, V., Vojtáš, P.: On top-k search with no random access using small memory. Accepted to ADBIS (2008). Preprint can be downloaded at `http://klud.ics.upjs.sk/~gursky/papers/2008adbis.pdf`.
9. Gurský, P., Horváth, T., Novotný, R., Vaneková, V., Vojtáš, P.: UPRE: User preference based search system. In IEEE/WIC/ACM Web Inteligence (2006)
10. Gurský, P., Vaneková, V., Pribolová, J.: Fuzzy User Preference Model for Top-k Search. In WCCI (2008)
11. Güntzer, U., Balke, W., Kiessling, W.: Towards efficient multi-feature queries in heterogeneous enviroments. In ITCC (2001)
12. Horváth T., Vojtáš, P.: Ordinal Classification with Monotonicity Constraints, In Proc. 6th Industrial Conference on Data Mining ICDM (2006)
13. Hristidis, V., Papakonstantinou, Y.: Algorithms and Applications for answering Ranked Queries using Ranked Views. VLDB Journal, Volume 13, Issue 1, (2004)

14. Ilyas, I., Aref, W., Elmagarmid, A.: Supporting top-k join queries in relational database. In VLDB (2003)
15. Ilyas, I., Shah, R., Aref, W.G., Vitter, J.S., Elmagarmid, A.K.: Rank-aware query optimization. In SIGMOD (2004)
16. Li, C., Chang, K., Ilyas, I., Song, S.: RankSQL: Query algebra and optimization for relational top-k queries. In SIGMOD (2005)
17. Ramakrishnan, R., Gherke, J.: Database management systems. Third edition, McGraw-Hill (2003)
18. Soliman, M.A., Ilyas, I.F., Chang, K.C.C.: Top-k Query Processing in Uncertain Databases. In Proc. ICDE (2007)
19. Re, Ch. , Dalvi, N.N., Suciu, D.: Efficient Top-k Query Evaluation on Probabilistic Data. In Proc. ICDE (2007)
20. Theobald, M., Schenkel, R., Weikum, G.: An Efficient and Versatile Query Engine for TopX Search. In VLDB (2005)
21. Yu, H., Hwang, S., Chang, K.: Enabling Soft Queries for Data Retrieval. Information Systems, Elsevier (2007)
22. Xin, D., Han, J, Chang, K.: Progressive and Selective Merge: Computing Top-K with Ad-Hoc Ranking Functions. In SIGMOD (2007)
23. Zhang, Z., Hwang, S., Chang, K., Wang, M., Lang, C., Chang, Y.: Boolean + Ranking: Querying a Database by K-Constrained Optimization. In SIGMOD (2006)