SUPPORTING SKYLINE QUERIES ON CATEGORICAL DATA IN WEB INFORMATION SYSTEMS

Wolf-Tilo Balke
Electrical Engineering and Computer Science
University of California, Berkeley, USA
balke@eecs.berkeley.edu

Ulrich Güntzer
Institut für Informatik
Universität Tübingen, Tübingen, Germany
guentzer@informatik.uni-tuebingen.de

Abstract
Skyline queries enable more intuitive querying, essential for e.g. e-commerce applications. However, the performance of query execution will drastically deteriorate, if categorical data is involved. Unfortunately most Web data tends to be of exactly that nature. In this paper we show how to remedy the gap in current skylining algorithms and adapt them for the nature of Web data. Our innovative algorithm minimizes the amount of expensive object accesses over the Internet and allows for progressive delivery of correct result objects at an early stage. These can already be syndicated with all necessary information and returned to the user while the search is still running. This also optimizes the use of available bandwidth and thus paves the road to efficient Web information systems.

Key Words
Databases and the Web, Web information systems, information retrieval, skyline queries, categorical data

1. Introduction

Today's information provisioning over the Internet poses more and more demanding problems. Besides the technical problems of accessing different Web sources and assembling the information with subsequent delivery to each end-user's device, the initial selection process is tantamount: selecting adequate information from a variety of Web repositories with respect to personal preferences still is a major challenge. Recent approaches towards intuitive information systems and the integration of user preferences basically lead to two different paradigms: 'top k' and 'skyline' queries. The *top k approach* uses a single general aggregation function to compensate between scores of different database objects and then delivers only a few top objects to the user. Typical compensation functions are e.g. weighted averages providing an overall best compromise between all aspects of user queries and the preferences given. Typical applications are economical choices e.g. a product's expected lifetime against its price.

Skyline queries on the other hand introduce the notion of dominated objects under Pareto optimality. An object is dominated and therefore should not be returned, if there are other objects that show at least the same score values in every query aspect and are strictly better in at least one. This leads to a choice of best matching objects in cases where compromises (or weightings) can hardly be specified in an intuitive or simple arithmetical way, e.g. comparing a product's security standards against its price. Both approaches have recently been adapted for the use in *Web information systems*, e.g. [4], [1]. Considering the nature of Web accesses, these algorithms focus on individual object accesses, not on working over a central index. Working directly on the objects' attributes, however, is of problematic complexity, see e.g. [7]. More efficient algorithms often make the indirection over numerical scorings like shown in [8]. Generally the desirability of each object with respect to a query predicate is mapped onto a score value between 0 and 1. These approaches have already been shown to be applicable for a variety of database applications or information services, e.g. for cooperative retrieval techniques ([2], [8], [3]) or content-based retrieval in multimedia systems ([5], [6]).

Database objects are always described by certain attributes. In Web applications, however, we very often experience a rather limited number of alternative values to choose from, i.e. limited *categories*. For instance, products may be available in different colors. But unlike in image indexing for multimedia databases, where virtually every combination of colors may occur in an image, the colors featured in a product database are often restricted to a couple of choices. For retrieval purposes this poses a difficult problem. Whereas in multimedia retrieval color features discriminate quite efficiently, there may be lots of objects with same colors in Web applications. Also for other categorical data, this situation often occurs. When booking a hotel over the Internet for instance, the room price may be quite discriminating, whereas the categories smoking/non-smoking only offer two choices and thus will -according to a user's preference- only divide the available rooms into those with score 1 and those with score 0. Though this problem does not really affect top k approaches, where the compensation may focus on the most discriminating query predicates first, e.g. [6], its effect on skyline queries is often disastrous. Here each attribute is considered incomparable and no compensation between different attributes is possible. Thus, if many objects are mapped onto the same score value, we have to explore large plateaus in some dimensions and thus access a vast number of objects over the Internet.

This paper will show how to deal with skyline queries in Web information systems efficiently, even if categorical data is involved. We will propose a novel algorithm that exploits all knowledge about interesting objects at an early time and thus will not need to explore vast numbers of irrelevant objects by expensive Web accesses. We will further prove theoretical results like the correctness of the result set, the efficient progressive delivery or the compatibility with previous approaches and demonstrate the practical usefulness and applicability of our approach.

2. Architectural Design and Related Work

We will now focus on the skylining problem and previous approaches. Moreover, we will explore the architectural design of Web information systems for efficient querying.

2.1 The Skyline Problem

The skyline retrieval paradigm has gotten a lot of attention recently as it proved especially useful for personalization issues. In contrast to pure economical transactions where each aspect of a query can be assigned a monetary value, personal user preferences often cannot be stated in a quantitative way and hence compensation between different aspects is difficult or even impossible. What does it mean in terms of the expected result set, if a query term is considered 0.65 times more useful as another? How would the result change, if a user had stated a relative importance of 0.53 instead? Therefore we will briefly revisit current skylining techniques and present an example for the use in Web information systems with limited or categorical domains.

The Skyline Problem: Given set $O := \{o_1, \ldots, o_N\}$ of N database objects, n score-functions $s_1, \ldots s_n$ with $s_i : O \rightarrow [0,1]$ and n sorted lists $S_1, \ldots, S_n$ containing all database objects and their respective score values using one of the score functions for each list; all lists are sorted descending by score values. Wanted is the subset P of all non-dominated objects in O, i.e. $\{o_i \in P \mid \neg\exists\ o_k \in O : (\ s_1(o_i) \leq s_1(o_k) \wedge \ldots \wedge s_n(o_i) \leq s_n(o_k) \wedge \exists\ q \in [1, \ldots, n]: s_q(o_i) < s_q(o_k))\}$

Basically skylining algorithms consist of three phases (see e.g. the algorithm in [1]):

- The first phase tries to find a common object in all sources by sorted accesses. Since this object dominates all objects having smaller scores all unseen objects can be discarded, only object already accessed or having at least the same score values as the common object ('hiding' behind it) can be part of the skyline.
- The second phase thus also has to access those objects with same score values (which in classical databases is mostly of no consequence, but for the limited categorical data in Web application usually leads to a vast number of additional accesses) and use random accesses to get all missing scores of any object seen.
- The third phase then compares the seen objects for domination and outputs the set of non-dominated objects as final skyline result.

Our concern in this paper is with the additional accesses within the second phase. We will present an improved algorithm that in most cases allows skipping the expensive additional accesses without risking the correctness of the final result set. Since categorical data with rather limited domains is quite common in Web applications, this is a worthwhile improvement. Let's consider a short application example:

Example (restaurant booking):
A business traveler wants to book a restaurant for the evening over the Internet. Due to his/her restricted budget for expenses he/she wants an inexpensive restaurant near to his/her hotel location showing a good rating. Local Internet sites like the respective yellow pages or centralized portals like restaurantrow.com, etc. together with independent ratings like Zagat.com will offer all the information our business traveler needs to complete his/her task. Thus for exploring the skyline of best choices our traveler can rely on the Web pages' lists of restaurants ordered by the respective scores for cheapest price, minimum distance to the hotel and best rating. But whereas price and distance are numerical domains and define a clear ordering, the Zagat rates restaurants into only thirty categories.

Exploring the skyline with today's algorithms like presented above would need to find a common object in all of the lists. Having found this object, previous approaches would also have to consider lots of objects showing the same score values in order not to miss a relevant object [1]. Though in numerical domains this hardly poses a problem, if objects can be sufficiently discriminated, in categorical domains, where we have to expect many objects with equal scores (plateaus), the effect is devastating. After we have found a sufficiently close and well-priced restaurant, for our termination condition it is no longer necessary to look at all other restaurants having the same Zagat rating, though they may be far more expensive and miles away. Moreover, our progressive delivery scheme will already have output all relevant objects. Keeping in mind that each access will mean a connection over the Internet, a strategy adapted to the needs of Internet applications, usually results in an essential improvement of the response time for our Web information system. In section 3 we will present our algorithm dealing efficiently with exactly these cases.

2.2 Web Information Systems Architecture

In Web applications, the objects' attributes are usually available through external Web-accessible form interfaces. There are usually two ways to access these interfaces: we can ask a Web source to list the best choices with respect to a certain aspect (e.g. restaurants sorted by their relative distance to the user) and then iterate over the result set one by one (also called a sorted access). In Web applications also the behaviour of immediately delivering a bulk of objects (top 10, next 10,…) is quite common. With only minor changes our skylining algorithm can obviously be adapted to this behaviour, thus in the following we will focus on sorted accesses of single objects. The second possibility of access is that we can simply ask a Web source after the attribute value of a specific object (e.g. the rating for a specific restaurant, also called a random access). To process a complex query over Web-accessible sources, we then have to interact with sources that export different interfaces and access capabilities.

For any query aspect there is a list of possible candidates that we iterate by sorted access. Usually each source will return a ranked list in decreasing order starting with the best choices and will either give the attribute values for each object or will even provide a scoring. Since for skyline queries we consider the exact score value as irrelevant, but only use it as a vehicle to get a ranking, we even do not have to normalize the scores over different query aspects. We just need the rank information together with some relative distance between the objects within each query aspect. For instance we can use the numbers given by the relative distance or the average price of the restaurant and just take the reciprocal value as scores to get highest scores for nearest or least expensive restaurants. Or, if a user had asked for a dinner around 20$ we can again map objects according to their price. Here a restaurant showing average prices of 22$ would get a higher score as restaurants with 15$ or 25$, which would in turn be mapped onto the same score value.

For categorical attributes that use numerical domains like the Zagat review we can also simply use the values as scores, in our case best-rated restaurants show a score of 30 declining with worse ratings. Even our query concerning a restaurant rated as 'good' is easily implemented by using the absolute difference of the actual restaurant rating and the representative value for good restaurants. For categorical data with non-numerical domains we have to assign an artificial score to each object. If for instance a user prefers a vegetarian restaurant, all those restaurants can be assigned a score of 1, while all other restaurants would get a score of 0 for this aspect. And even if there are more subtle differences, usually a suitable metric can be found to order objects: if for example a user prefers Indian cuisine, Thai restaurants may still be assigned a higher score than e.g. Chinese or Italian restaurants. Generally speaking we can always rely on whatever information the Web source used to list its objects. Hence, we can interact with various autonomous sources and repeatedly query them for a potentially large set of candidate objects.

Figure 1 shows a rough sketch of our architecture. Central part is an application server running a query, skyline and delivery engine. The user's query is split and for each aspect the respective Web source is queried by sorted and/or random access. Since we have to register the available Web sources with our information system anyway, we can always decide for an adequate scoring. The objects are then scored either directly using the scoring of the Web source or introducing a simple scoring to make objects comparable within each query aspect (where objects of the same desirability are mapped onto same score values). The skyline engine calculates the optimal set of objects, i.e. the skyline, and the result is subsequently delivered to the user. For the delivery different stylesheets (e.g. using XSLT [9], [2]) can be provided adapting the result to the users client device (e.g. for mobile access).

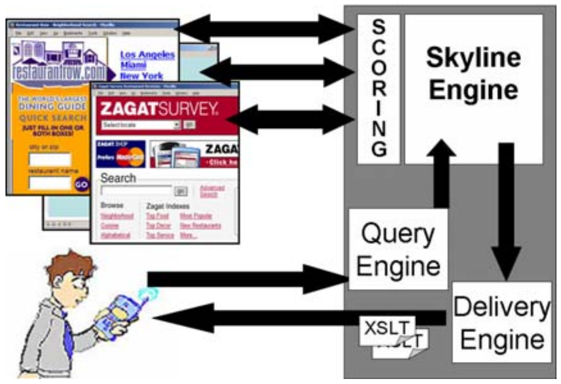


Figure 1: Architecture for skylining over Web sources

3. An Algorithm for Categorical Data

Considering the limited discriminative power of some of the data in Web applications can we improve the general behavior of skylining? Let us first consider an alternative condition, which allows us to discard unseen objects and then show that, though the condition is somewhat stricter than the original one, it will help us to deal with plateaus of equal scores. We can discard all unseen objects in any case where we have at least one object that has a better score than any unseen object, i.e. in terms of Pareto optimality its scores are larger or equal, but in at least one aspect strictly larger than any unseen objects. Since the scores for unseen objects due to the sorting of each Web source's score lists are upper bounded by the minimum score accessed by iterating the lists (sorted access), we can monitor our condition constantly. But let us first prove that our stricter condition really allows discarding all unseen objects.

Lemma 1: Correctness for discarding unseen objects
Let $p_1,\ldots, p_n$ be the minimum scores seen by sorted access for each of the n query aspects. If an object o fulfilling $s_i(o) \geq p_i$ $(1 \leq i \leq n)$ and $s_i(o) > p_i$ for at least one i has been accessed, we can safely discard all unseen objects.
Proof:
Let o be our object dominating the vector of actual minimal scores and u be any object yet unseen. Since we have

iterated the lists for any query aspect one by one and all lists are sorted, the highest possible scores for any yet unseen object are give by the minimum scores $p_1, \ldots, p_n$. Therefore we have $s_i(o) \geq p_i \geq s_i(u)$ and for at least one i: $s_i(o) > p_i \geq s_i(u)$. Thus also the definition of domination with respect to Pareto optimality between o and u holds and we have an object that dominates any yet unseen object. Hence, no unseen object can be part of the skyline and we can safely discard all unseen objects. ■

Please note that in spite of our condition being stricter, we have not insisted on finding a common object. Thus instead of looking for a common object in all lists delivered by the Web sources, our algorithm will immediately do all necessary random accesses to assess each accessed object's quality. Moreover, it will use the minimal scores seen by sorted accesses as a threshold, so that our condition already can become true when entering a plateau, not only after finishing it. But we can do even more; instead of waiting for our condition to become true and then do all necessary comparisons to remove dominated objects afterwards, we will test objects for domination already at an early stage using our notion of score plateaus. The following lemma will show that we can recognize objects of the skyline as soon as accesses on a score plateau have been finished in some list. If an object has been seen in list S_i, it suffices to compare it with those objects, which have better or equal scores s_i. Exploiting this idea, we can output the skyline objects on the fly beginning at an early stage and thereby improve response times drastically.

Lemma 2: Successive output of skyline objects
If we have finished a score plateau by sorted access in any list (i.e. accessed an object, whose score is strictly smaller than the score previously accessed in this list), any object of this plateau, which after pairwise comparisons for domination with all other objects occurring in the plateau and tests for dominations with all objects that occurred earlier in the same list is still non-dominated, is definitely part of the final skyline and can already be output.
Proof:
We have to show that an object always can only be dominated by objects in the same plateau or those having occurred earlier in the same list. Assume we have just finished accessing a plateau in list S_x and let o_x be any object accessed in that plateau. According to the definition of Pareto optimality an object has to have higher or at least equal score in *all of the* lists to dominate another object. Hence, o_x can never be dominated by an object having a smaller score value in S_x. Since we have finished o_x's score plateau by accessing an object having a strictly smaller score with respect to S_x, due to the sorting of S_x all objects still unseen in S_x cannot dominate o_x. Thus, we only have to test o_x for domination against all objects already having occurred in S_x and if it is not dominated, it is part of the final skyline. Since o_x has a smaller score than any object seen earlier in S_x and not on o_x's score plateau, o_x itself cannot possibly dominate these objects, but only be dominated by them. ■

Finally, the next lemma will show that at the time our termination condition becomes true, we can not only discard all unseen (lemma 1), but also all relevant objects have already been output (lemma 2) and we can simply stop the algorithm.

Lemma 3: Completeness of output skyline objects
If any seen object dominates the minimum scores in each list, and objects have been output progressively like stated before, the *entire* skyline has already been output.
Proof:
Lemma 2 shows that skyline objects are output, after a plateau in any list has been entirely accessed. So we have to show that no object which has not yet been output can belong to the skyline. Let p_i be the minimum scores seen by sorted access in each list. Since according to lemma 2 we have output all relevant objects w with $s_i(w) > p_i$ for some i, $1 \leq i \leq n$, we have to check that no object o with $s_i(o) \leq p_i$ for *all* i can belong to the skyline. However, since we have seen an object q dominating the minimum scores, lemma 1 states that for all i we know $s_i(o) \leq p_i \leq s_i(q)$ and there is an index j for which $s_j(o) \leq p_j < s_j(q)$. Thus object o is always dominated by object q and can never be part of the skyline ■

So, given a query consisting of n different aspects, we will now present the algorithm of how to find the skyline objects efficiently, even if the data in the underlying sources is of categorical nature:

Algorithm: Skylining for Web Applications

1. For each aspect of the query access one of the registered Web sources and pose the respective query part to get a list of best objects. If for any aspect no Web source should yet be registered, ask the user either to drop that part, or to register a respective Web source.
2. Using a round robin strategy perform a sorted access on a Web source and get an object either together with its score (if provided by the Web source) or assign a score to the object (considering either its rank number or its attribute value relative to the query).
3. If the object's score is strictly smaller than the one previously accessed by sorted access on the same list, compare all objects of the list's previous score plateau pairwise and check for domination with objects that occurred earlier in the same list. Discard all dominated objects from the actual list (and mark them as already discarded in case they also will occur in other lists later on) and output all non-dominated objects of the plateau as skyline objects.
4. For each new object accessed in step 2 do random accesses on the other Web sources to get all score values for the object.
5. Check if there is an object among the objects seen, such that its scores are larger or equal the current minimum scores seen by sorted access in each Web source and strictly larger for at least one score value. If this is not the case, go back to step 2, else terminate the algorithm.

Step 3 performs the successive output of objects according to lemma 2. Please note that after step 4 all relevant information for the new object under consideration has been accessed and transferred to the application server running our skyline engine, so the rest can be done on our application server. Since the termination of the algorithm is obvious and we have proved its correctness in Lemma 1, we will now focus on its performance.

4. Evaluation and Use Case Study

4.1 Performance of Skylining

In this section we will investigate the performance of our approach and show its strengths over previous approaches for Web applications. If no categorical data is involved, i.e. the range of scores discriminates objects well enough, our stricter condition behaves like previous approaches:

Lemma 4: Relationship to previous approaches
If any seen object dominates the minimum scores (like in lemma 1), the object has either already been seen by sorted access in all sources or for the sources, where it has not yet been seen, is within a plateau of objects having equal scores to the respective minimum score accessed.
Proof:
Let o be our dominating object and $p_1, \ldots, p_n$ the minimum scores seen by sorted access in each Web source. Since o dominates the minimum scores, due to the definition of Pareto optimality we have $s_i(o) \geq p_i$. Hence we can divide the set of indices in those, for which $s_i(o) > p_i$, and those, for which $s_i(o) = p_i$. In the first case due to the sorting of the lists the object o must have been accessed already when iterating down to p_i. In the second case the object o might not have been accessed yet, but since its score is equal to the current minimum, it has at least to be part of the current plateau like claimed above. ■

Lemma 4 shows that, though we may not gain much in normal database retrieval without categorical data (in the specific case of injective score functions our condition is actually equivalent to the previous approaches), we still can expect an essentially improved performance, if large score plateaus occur. Since that is the case exactly in Web applications, we will no consider a short example to show how the algorithm deals with categorical data. Consider our example from above. We have three Web sources that score the restaurants of a certain region (e.g. the San Francisco Bay Area) according to their closeness S_1, the average dinner price S_2 and the rating S_3. Let us for simplicity name the restaurants R_i and assume scores normalized to [0,1] in all of the different lists. Please note that due to the Bay Area containing a lot of restaurants rated as 'good' (Zagat score 15-20) we are bound to experience a (probably large) plateau of top scored restaurants in S_3 independently of their respective location and price. The next table gives some top ranked objects from each list:

	S_1		S_2		S_3	
rank	oid	score	oid	score	oid	Score
1	R_1	0.99	R_2	0.9	R_3	1.0
2	R_4	0.94	R_5	0.85	R_6	1.0
3	R_7	0.85	R_8	0.83	R_9	1.0
4	R_{10}	0.84	…	…	…	…

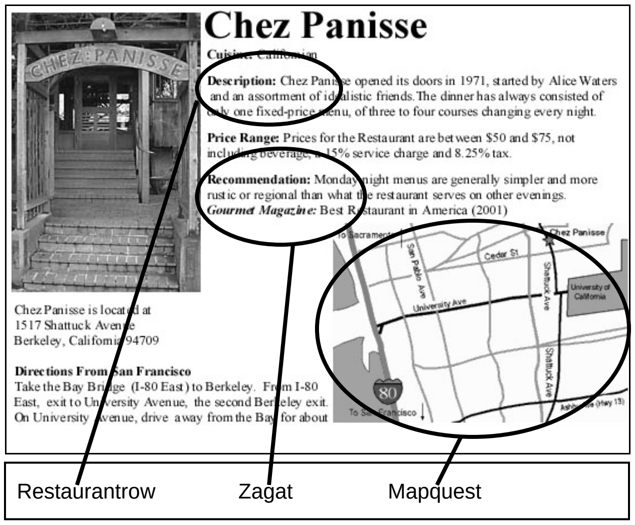
As stated in our algorithm after the initialization of step 1 we will start sorted accesses on all lists in a round robin fashion (shown bold in the table below). We then immediately do random accesses on the remaining scores of each new object and mark the current minima seen by sorted access in each list. For instance doing a sorted access on S_1 we access R_1 with a score of 0.99. Doing the random accesses for R_1 for s_2 and s_3 we get scores of 0.6 and 0.8 respectively. Due to the sorted access, the minimum score in S_1 is now decreased to 0.99, and so on.

oid	score S_1	score S_2	score S_3	current minima
R_1	**0.99**	0.6	0.8	(0.99, 1.0, 1.0)
R_2	0.7	**0.9**	0.1	(0.99, 0.9, 1.0)
R_3	0.45	0.83	**1.0**	(0.99, 0.9, 1.0)
R_4	**0.94**	0.83	1.0	(0.94, 0.9, 1.0)
R_5	0.76	**0.85**	0.8	(0.94, 0.85, 1.0)
R_6	0.5	0.6	**1.0**	(0.94, 0.85, 1.0)
R_7	**0.85**	0.55	0.2	(0.85, 0.85, 1.0)
R_8	0.6	**0.83**	0.7	(0.85, 0.83, 1.0)

After accessing R_4 we see that R_1 (previous score plateau) cannot be dominated by any other object and thus is output. The same happens after the access of R_5, where R_2 can be immediately output. Note that the access on R_6 does not lead to any outputs, because of the large score plateau in S_3. The access on R_7 will output R_4 and after accessing R_8, not only can we output R_5, but for the first time have an object dominating the minimum scores, namely R4. Please note that R4 did not yet occur in all lists, though it is part of the current minimum score plateaus in S_2 and S_3, but there could very well be plenty of other objects with e.g. s_3=1.0, before R_4 eventually shows up in both lists Following previous approaches we would have to clear these plateaus by expensive accesses over the Web to find our object in all lists. Using our approach with the stricter condition, we know at this early stage that we can already stop accesses and the complete skyline has already been delivered on the fly. As we can see, we save an essential amount of useless object accesses depending on the number and relative length of score plateaus in the different Web sources.

4.2 Customized Delivery

Having found the skyline and given some information about the user device we can now customize the delivery of the result objects. Our technique of random access together with the knowledge of each Web source allows us not only to get scorings for objects, but also somewhat richer information as for example a brief description or sometimes even a photo of each selected restaurant from restaurantrow, the detailed scoring and comments of referees from the Zagat review or helpful things like a map and route description from e.g. mapquest. As pointed out in section 2.2. using common XML techniques (like XSLT stylesheets, for a detailed implementation see e.g. [9] or [2]), we can not only customize the specific format in which each client device can process the information, but also syndicate the content according to specific users' preferences. For instance we could deliver a certain preferred layout or e.g. for limited mobile client devices scale down or remove images. An example of our syndicated delivery is shown in figure 2.



Chez Panisse

Cuisine: Californian

Description: Chez Panisse opened its doors in 1971, started by Alice Waters and an assortment of idealistic friends. The dinner has always consisted of only one fixed-price menu, of three to four courses changing every night.

Price Range: Prices for the Restaurant are between $50 and $75, not including beverage, a 15% service charge and 8.25% tax.

Recommendation: Monday night menus are generally simpler and more rustic or regional than what the restaurant serves on other evenings.
Gourmet Magazine: Best Restaurant in America (2001)

Chez Panisse is located at
1517 Shattuck Avenue
Berkeley, California 94709

Directions From San Francisco
Take the Bay Bridge (I-80 East) to Berkeley. From I-80 East, exit to University Avenue, the second Berkeley exit. On University Avenue, drive away from the Bay for about

Figure 2: Syndication for result delivery

Since our algorithm allows for successive output of skyline objects, we can generate such result pages at the earliest possible moment and immediately deliver them. Thus the user can start to deal with the first objects while the retrieval algorithm is still running. Especially in mobile scenarios with limited bandwidths and slow connections over the Internet this results in a major advantage.

5. Summary and Outlook

We have presented an innovative approach to enable efficient processing of skyline queries in Web information systems. Our approach focuses on Web applications that are usually characterized by a large amount of categorical data. That means that the values for certain query predicates are often simple categories in rather limited domains. Unlike previous approaches we have proven our algorithm to always deliver the correct result taking the nature of Web data into account: instead of always sifting through the entire categories our novel approach closely focuses on the score information and discards unnecessary objects at an early stage. Moreover, due to its progressive delivery scheme the user is provided with first result objects, while the algorithm is still running. This essentially improves retrieval time and optimizes the use of bandwidths. Moreover, we focused on a suitable architecture enabling an early syndication of relevant content and its adaptation to the needs of various client devices.

Our future work will focus on integrating skyline queries even more closely with other retrieval paradigms. Given that query predicates most of the time will not be only either compensatory or entirely incomparable, we will investigate ways towards multi-objective retrieval algorithms where any mixture between compensation and Pareto optimal retrieval can be processed. Though our algorithm presented here is a prerequisite towards enhancing the retrieval capabilities of (distributed) Web databases, the ability to process queries closely respecting more complex user preferences and non-numerical notions of utilities is still a major challenge.

References

[1] W.-T. Balke, U. Güntzer, J. Zheng. Efficient Distributed Skylining for Web Information Systems. In *Int. Conf. on Extending Database Technology (EDBT'04)*, LNCS 2992, Heraklion, Crete, Greece, 2004

[2] W.-T. Balke, W. Kießling, C. Unbehend. Personalized Services for Mobile Route Planning: A Demonstration. In *Int. Conf. on Data Engineering (ICDE 2003)*, Bangalore, India, 2003

[3] S. Börzsönyi, D. Kossmann, K. Stocker. The Skyline Operator. In *Int. Conf. on Data Engineering (ICDE'01)*, Heidelberg, Germany, 2001

[4] N. Bruno, L. Gravano, A. Marian. Evaluating Top-k Queries over Web-Accessible Databases. In *Int. Conf. on Data Engineering (ICDE'02)*, San Jose, USA, 2002

[5] R. Fagin, A. Lotem, M. Naor. Optimal Aggregation Algorithms for Middleware. *ACM Symp. on Principles of Database Systems (PODS'01)*, Santa Barbara, USA, 2001

[6] U. Güntzer, W.-T. Balke, W. Kießling. Optimizing Multi-Feature Queries for Image Databases. In *Int. Conf. on Very Large Databases (VLDB'00)*, Cairo, Egypt, 2000

[7] W. Kießling, G. Köstler. Preference SQL - Design, Implementation, Experiences. In *Int. Conf. on Very Large Databases (VLDB'02)*, Hong Kong, China, 2002

[8] A. Motro. VAGUE: A User Interface to Relational Databases that Permits Vague Queries. ACM Trans. Information Systems. 6(3): 187-214, 1988

[9] M. Wagner, W.-T. Balke W. Kießling. An XML-based Multimedia Middleware for Mobile Online Auctions. In *Enterprise Information Systems III*, Kluwer Academic Publishers, The Netherlands, 2002