Evaluating Top-k Skyline Queries over Relational Databases

Carmen Brando, Marlene Goncalves, and Vanessa González

Universidad Simón Bolívar, Departamento de Computación y TI, Apartado 89000
Caracas 1080-A, Venezuela
{carmen,mgoncalves,vgonzalez}@ldc.usb.ve

Abstract. Two main languages have been defined to allow users to express their preference criteria: Top-k and Skyline. Top-k ranks the top k tuples in terms of a user-defined score function while Skyline identifies non-dominated tuples, i.e. such tuples that does not exists a better one in all user criteria. A third language, Top-k Skyline, integrates them. One of the drawbacks of relational engines is that they do not understand the notion of preferences. However, some solutions for Skyline and Top-k queries have been integrated into relational engines. The solutions implemented outside the core query engine have lost the advantages of true integration with other basic database query types. To the best of our knowledge, none of the existing engines supports Top-k Skyline queries. In this work, we propose two evaluation algorithms for Top-k Skyline which were implemented in PostgreSQL, and we report initial experimental results that show their properties.

1 Introduction

Currently, many applications, such as decision support or customer information systems, may take advantage of preference queries in order to find best answers. Thus, in the case of a decision support system, a possible query could be to determine the best customers who have made many purchases and have little or no complaints. In this example, two user criteria have been defined: many purchases and few complaints. In a SQL query, these criteria may be specified using an ORDER BY clause where the number of purchases is sorted ascendingly, and the number of complaints is sorted descendingly. Additionally, the relational engine must sort the answers and the user must discard all irrelevant data. Thus, the number of answers to be analyzed by the user might be enormous and declarative query languages like SQL might not be the most suitable ones to evaluate user preferences.

Hence, SQL was extended to consider user criteria and for this purpose, two preference languages were defined: Top-k and Skyline. The first, produces the top k tuples based on a user-defined score function that induces a total order. The second, identifies non-dominated tuples based on a multicriteria function defined by the user. A tuple dominates another one if it is as good or better than the other in all criteria and better in at least one criterion. The multicriteria function, that seeks to maximize or minimize criteria, induces a partially ordered set and, in consequence, there are several optimal answers.

Top-k and Skyline are two different ways to solve a preference query. On one hand, Top-k requires that a score or weight function be defined and this might not be natural for all users. On the other hand, Skyline retrieves tuples where all criteria are equally important and a score function cannot be defined. However, the number of Skyline answers may be smaller than required by the user, who needs at least k. Given that Skyline does not discriminate its answers in the same way as Top-k does and that Top-k does not allow to solve queries where a score function can not be defined a priori, Top-k Skyline was defined as a unified language to integrate them. Top-k Skyline allows to get exactly the top k from a partially ordered set stratified into subsets of non-dominated tuples. The idea is to partition the set into subsets (strata) consisting of non-dominated tuples and to produce the Top-k of these partitions.

Many solutions have been given for Skyline and Top-k, being the best those that integrate them into a relational engine because these avoid performance degradation. Performance improvement is due to the query evaluation being done inside the database system, instead of retrieving tuple by tuple in a midleware application and then, computing the Skyline or Top-k. Regarding that these solutions have demonstrated to be successful, we propose to integrate Top-k Skyline queries into a relational engine. To the best of our knowledge, none of the existing engines supports Top-k Skyline queries. In this work, we propose two evaluation algorithms for Top-k Skyline which were implemented in PostgreSQL, and we report initial experimental results that show their properties.

Finally, this paper comprises five sections. In Section 2, we introduce a motivating example. In Section 3, we briefly describe Skyline and Top-k as Related Work. In Section 4, we present the definition for Top-k Skyline. In section 5, we describe our solutions for Top-k Skyline queries. In Section 6, we report our initial experimental results. And, finally, in Section 7, the concluding remarks and future work are pointed out.

2 Motivating Example

Lets consider that a government program runs a scholarship contest. For this purpose the applicants must submit their resumes, providing information on their academic and professional performance and also an application form with data concerning their annual income. The summarized information is organized in the relational table:

$$\textit{Candidate}(\underline{\textit{name}}, \textit{degree}, \textit{GPA}, \textit{publications}, \textit{income}),$$

where *degree* refers to the last academic degree achieved, *GPA* to the corresponding GPA in a scale from 1 to 5, *publication* represents the number of scientific publications writen, and *income* relates to the annual income in euros (€). Table 1 shows some tuples for this relation.

The resulting candidates with best academic merits must include the ones with the highest degree, the greater number of publications and the best GPA. Each

Table 1. Candidate relation

name	degree	GPA	publications	income
Sue Smith	MsC	3.6	3	25,000
Hansel Bauer	PhD	4.0	8	30,000
Jamal Jones	MsC	4.2	2	25,000
Kim Tsu	PhD	4.3	3	27,500
María Martínez	BEng	4.5	1	33,000

of these criteria are equally important, making score functions inappropriate for this selection. All the applicants that satisfy these criteria should make finalists. Thus, a candidate can only be chosen to win a scholarship if and only if there is no other candidate with higher degree, number of publications and GPA. Thus, the finalists are Hansel Bauer, Kim Tsu and María Martínez. Sue Smith and Jamal Jones, are disregarded given they have lower degrees, GPAs and number of publications than any other.

Additionally, the government program might have to choose a limited number of winners, say $k = 2$. Under these new circumstances, it is necessary to define a new criterion to discriminate between the finalists in order to only yield k. Therefore, the awarded students will be the ones that better qualify given a score function defined by the panel of judges, e.g. a score function defined on their income to aid the candidates with the lowest ones. From the finalists, the judges select the new scholarship winners: Hansel Bauer and Kim Tsu, because María Martínez has higher income than the other two.

We have now intuitively evaluated a preference query. On one side, *Skyline* determined the finalists without restricting the result to exactly k tuples. On the other hand, *Top-k* returned the k winners from the finalists with the lowest income. If we were only to apply the score function to the candidates, without the previous *Skyline*, results could differ, because Top-k might favor some candidates that could be dominated by others and these are considered false results under Skyline criteria. Note that if we apply only Skyline the results could be incomplete when the number of Skyline answers is lower than k. Neither Skyline nor Top-k can solve this kind of problem, it is then necessary a solution such as Top-k Skyline for the problem of selecting student scholarship winners. With Top-k Skyline, the answers are guaranteed to be sound and complete for this case.

3 Related Work

Several algorithms have been proposed in order to identify the Skyline in relational database systems: Divide and Conquer extension, Block-Nested-Loops (BNL), Sort-Filter-Skyline (SFS) and LESS (Linear Elimination Sort for Skyline). All of them [9] scan the entire table and compute the Skyline. Also, progressive Skyline algorithms are introduced in [19], [14], [18], meanwhile algorithms for distributed systems are presented in [2], [3], [15], [12].

On the other hand, Top-k solutions try to avoid probing a user-defined score function on all of the tuples and to stop as early as possible. One of the first approaches was done by Carey and Kossman [5], [6], they proposed a new SQL operator called STOP AFTER k that indicates how many objects are required. Then, some algorithms were defined that focus on the problem of minimizing the number of probes [8], [17], [1], [4], [7], [16], [13].

Recently, solutions for the combination of Skyline and Top-k have been defined [3], [10], [15]. In general, these solutions calculate the first stratum or Skyline with some sort of post-processing. None of these solutions identify the k best answers considering situations like the one described in the previous section.

4 Top-k Skyline

Given a set $T = \{t_1, \dots, t_n\}$ of database tuples, where each tuple t_i is characterized by p attributes $(A_1, \dots, A_p)$; r score functions $s_1, \dots, s_r$ defined over some of those attributes, where $s_i : O \rightarrow [0, 1]$; a combined score function f defined over combinations of the score functions $s_1, \dots, s_r$ that induces a total order of the tuples in T; and a multicriteria function m defined also over some of the score functions $s_1, \dots, s_r$, which induces a partial order of the tuples in T. We define Strata according to multicriteria function m through the recursion presented in Definition 1. For simplicity, we suppose that the scores related to the multicriteria function are maximized.

Definition 1a (Base Case: First Stratum R_1 or Skyline)

$$R_1 = \left\{ \begin{array}{l} t_i \in T / \neg\exists t_j \in T : (s_1(t_i) \leq s_1(t_j) \wedge \cdots \wedge s_r(t_i) \leq s_r(t_j) \\ \qquad\qquad\qquad\qquad \wedge \exists q \in \{1, ..., r\} : s_q(t_i) < s_q(t_j)) \end{array} \right\}$$

Definition 1b (Inductive Case: Stratum R_i)

$$R_i = \left\{ \begin{array}{l} t_l \in T / t_l \notin R_{i-1} \wedge \neg\exists t_u \in (T - \cup_{j=1}^{i-1} R_j) : \\ \qquad\qquad\qquad\qquad (s_1(t_l) \leq s_1(t_u) \wedge \cdots \wedge s_r(t_l) \leq s_r(t_u) \\ \qquad\qquad\qquad\qquad \wedge \exists q \in \{1, ..., r\} : s_q(t_l) < s_q(t_u)) \end{array} \right\}$$

These two definitions establish that the tuples comprising each stratum will only be dominated by others in strata prior to their own. An efficient solution to the Top-k Skyline problem should avoid building all strata. In fact, it should only create the necessary ones. A Stratum R_i of $R =< R_1, \dots, R_n >$ is necessary if and only if exist strata $R_1, \dots, R_i, \dots, R_v$, where $v \leq n$ and v is the minimum number of strata that satisfy $| \cup_{i=1}^{v} R_i| \geq k$. On the other hand, it should only include the tuples from the last necessary stratum with the highest score values until there are k tuples in the answer. The answer for a relational Top-k Skyline query includes all the tuples in the strata $R_1, \dots, R_{v-1}$ (Previous Necessary Strata), plus those in R_v (Last Necessary Stratum) with the highest scores in f required to complete k tuples. Thus, we define Previous Necessary Strata and Last Necessary Stratum in Definition 2.

Definition 2.1 (Previous Necessary Strata R_{ps})

$$R_{ps} = \{\cup_{i=1}^{v-1} R_i / | \cup_{i=1}^{v} R_i| \geq k > | \cup_{i=1}^{v-1} R_i|\}$$

Definition 2.2 (Last Necessary Stratum R_{lt})

$$R_{lt} = \{R_v / | \cup_{i=1}^{v} R_i| \geq k > | \cup_{i=1}^{v-1} R_i|\}$$

Finally, the conditions to be satisfied by the answers of a relational Top-k Skyline query are given in Definition 3.

Definition 3 (Relational Top-k Skyline TKS)

$$TKS = \left\{ t_i \in T / t_i \in R_{ps} \vee (t_i \in R_{lt} \wedge \neg \exists^{k-|R_{ps}|} t_j \in R_{lt} : (f(t_j) > f(t_i))) \right\}$$

5 Our Proposed Solutions for Top-k Skyline

Two kinds of Skyline algorithms have been proposed in relational databases. The first kind scans the entire input and the second does not necessarily scans all the tuples, because it is index-based. In this work, we do not regard index-based algorithms, primarily considering that efficient access to data does not affect performance of a Skyline query as much as the multicriteria function evaluations do; secondly, although the Skyline can be precomputed through indexes, the Skyline of a set of attributes can not be calculated from the skylines of subsets of attributes, and viceversa; thirdly, the preceding option may be invalid when a new tuple is inserted into the database such that it dominates some tuples from the index and, in consequence, the entire index must be recalculated; finally, Skyline might be calculated from a set of materialized tuples -instead of base tables- and the index can only be applied to base tables.

Block-Nested-Loops (BNL), Sort-Filter-Skyline (SFS) and LESS (Linear Elimination Sort for Skyline) are three relevant algorithms for Skyline computation in relational databases because of their performance [9]. The BNL algorithm scans the entire table while it maintains a window of non-dominated tuples, which could be replaced by any other tuple that is seen later on. In Algorithm 1, we present a BNL extension for Top-k Skyline queries in a relational setting. The Extended Block-Nested-Loops (EBNL) algorithm receives a relation R, a multicriteria function m and a combined score function f as input and produces the Top-k Skyline tuples in terms of m and f. The iteration corresponding to steps 5 through 30, calculates the necessary strata. Meanwhile, the one from 7 to 27 corresponds to BNL, with the difference that here, dominated tuples are not discarded, instead, they are stored into a temporary file $R1$, so they can be used to determine the next stratum, if necessary. If the first stratum contains k tuples (condition verified on step 5) the algorithm stops, else the input set is replaced with the temporary file R_1 (step 29) and hence, the necessary strata are built one at a time, without partitioning all the data, until there are k or more Top-k Skyline tuples. In steps 7 through 27, BNL algorithm is executed without discarding dominated tuples. When a tuple p is read from the input set (steps 9-21), p is compared against all the tuples in the window (steps 10-19) and: if p is dominated by any tuple in the window, then p is inserted into a temporal file R_1; else, if p dominates any tuples in the window, these dominated tuples are removed from the window and inserted into a temporary file R_1, and p is inserted into the window; and if p is non-dominated, then it is inserted into the window. If, in any of the mentioned situations where p must enter the window, there is not enough room in it, p is inserted into another temporary file R_2 in order to be processed in the next iteration of the algorithm. Finally, if the temporary file R_2 is not empty, the input set is replaced with it to resume computing the current stratum (steps 22-24).

Algorithm 1. Extended Block-Nested-Loops Algorithm

```
1: INPUT: $R$: relation; $m$: multicriteria function; $f$: combined score function.
2: OUTPUT: Top-k Skyline tuples.
3: Initialize $i \leftarrow 1; count \leftarrow 0$;
4: Create a window $w$ of incomparable tuples in main memory;
5: while $count < k$ and exist tuples in $R$ do
6:    Initialize $P_i \leftarrow \emptyset; continue \leftarrow true$
7:    while (continue) do
8:       Get the first tuple $t$ from $R$;
9:       while exist tuples in $R$ do
10:          if some tuple $t_1$ from $w$ dominates $t$ then
11:             $t$ is inserted into the temporal table $R_1$;
12:          else if $t$ dominates some tuples from $w$ then
13:             insert $t$ into $w$;
14:             delete dominated tuples from $w$ and insert them into $R_1$;
15:          else if no tuple $t_1$ from $w$ dominates $t$ and there is enough room in $w$ then
16:             $t$ is inserted into the window $w$;
17:          else if no $t_1$ from $w$ dominates $t$ and there is not enough room in $w$ then
18:             $t$ is inserted into a temporal table $R_2$;
19:          end if
20:          Get the next tuple $t$ from $R$;
21:       end while
22:       if exist tuples in $R_2$ then
23:          $R \leftarrow R_2$;
24:       else
25:          $continue \leftarrow false$;
26:       end if
27:    end while
28:    Evaluate $f$ for all tuples in $w$; copy tuples from $w$ to $P_i$;
29:    $count \leftarrow count + size(P_i); i \leftarrow i + 1; R \leftarrow R_1$;
30: end while
31: return Top-k Skyline tuples;
```

Similarly, we extended SFS for Top-k Skyline computing in relational database contexts. SFS could be regarded as a BNL variant, since it only requires a previous sorting step based on a topological order compatible with the Skyline criteria and does not need window tuple replacement like BNL does (steps 12-14 of Algorithm 1), because of the mentioned initial topological sort; for further details see [9]. Steps 10 through 19 of Algorithm 1 are replaced by steps 1 through 7 of Algorithm 2. Additionally, the first statement that SFS must execute is a sorting on the input, based on the topological order compatible with the multicriteria function.

Algorithm 2. A portion of Extended Sort-Filter-Skyline Algorithm

```
1: if some tuple $t_1$ from $w$ dominates $t$ then
2:     $t$ is inserted into the temporal table $R_1$;
3: else if no tuple $t_1$ from $w$ dominates $t$ and there is enough room in $w$ then
4:     $t$ is inserted into the window $w$;
5: else if no $t_1$ from $w$ dominates $t$ and there is not enough room in $w$ then
6:     $t$ is inserted into a temporal table $R_2$;
7: end if
```

Finally, Algorithm LESS initially sorts tuples as SFS does, but presents two improvements over it: in the first ordering phase, it uses an elimination-filter window to discard dominated tuples quickly and it combines the last phase of the sort algorithm with the Skyline filter phase of SFS to eliminate remaining dominated tuples. For our Top-k Skyline problem, LESS is not easily extensible for the stratification of data because the Skyline order would only be profited by the first stratum, while the tuples that are inserted into a temporary file to be used in the determination of the following strata become randomly ordered, and multiple sorting phases would be necessary to discard the dominated tuples of each stratum.

6 Experimental Study

Our experimental study was performed on **PostgreSQL 8.1.4**. We have extended PostgreSQL to include a logical operator for Top-K Skyline. This operator has two physical implementations, namely EBNL and ESFS, and is evaluated after all the relational operators, i.e. scan, join, sort, etc. This is due to the semantics of Top-k Skyline queries. Since our PostgreSQL extension does not optimize these queries, only minor changes to the parser, rewriter and optimizer were necessary in order to integrate this new operator.

This study consisted of experiments running over a relational table with 25,000 tuples. The table contains an identifier and six real number columns that represent the scores. Values of numeric columns vary from 0 to 1. The attribute values were generated following a uniform data distribution.

The algorithms were executed on a Sun Fire V240 with 2 UltraSPARC IIIi processors of 1503MHz, 2 GB of memory and an Ultra160 SCSI disk of 146 GB, running SunOS 5.10.

Ten queries were generated randomly, characterized by the following properties: (a) there is only one table in the FROM clause; (b) the attributes in the multicriteria function and combined score function are chosen randomly from the attributes of the table using a uniform distribution; (c) the optimization type for each attribute of the multicriteria function is selected randomly considering MIN and MAX directives; (d) the number of attributes of the multicriteria function is two, four and six; and (e) k corresponds to 3% of the data size.

In this experimental study, the number of multicriteria function evaluations and the time taken by each algorithm were measured. Figure 1 reports the average of multicriteria function evaluations (x10^{-6}) and average time (in x10^{-4} miliseconds) used by each algorithm. Average time is the average of ten queries ran against the table with data generated according to a uniform distribution.

We have observed that EBNL requires more multicriteria function evaluations than ESFS. This could be attributed to the benefits gained through sorting as a first step in ESFS. Besides, EBNL is more efficient on smaller Skylines and the strata of the experimental queries were not small. Also, EBNL behavior deteriorates as the dimensions grow, because strata size increases. On the other hand, ESFS remains stable and it is not affected by strata size.

Finally, EBNL requires more time than ESFS does. This is because of the number of multicriteria function evaluations. Additionaly, ESFS time is not affected by the sorting phase because it reduces the number of multicriteria function evaluations.

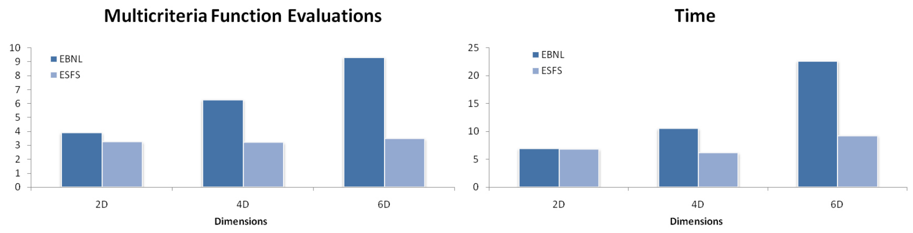

Fig. 1. Uniform data

Similarly, we ran experiments over a database that contains real web data. We created a table with information from Zagat Survey Guides [20]. This study consisted of experiments running over a relational table with, approximately, 16,876 tuples. The table contained an identifier and five real number columns that represent the scores. Values of numeric columns vary from 0 to 30. Any given column may have duplicate values.

Five queries were generated randomly, characterized by the following properties: (a) there is only one table in the FROM clause; (b) the attributes in the multicriteria function and combined score function are chosen randomly from the attributes of the table using a uniform distribution; (c) the optimization type for each attribute of the multicriteria function is selected randomly considering

MIN and MAX directives; (d) the number of attributes of the multicriteria function is four; (e) k corresponds to 1%, 3% and 5% of the data size.

For this study, the number of multicriteria function evaluations (x10^{-6}) and the time (x10^{-3} miliseconds) taken by each algorithm were measured. Figure 2 reports the results of the experiments for the real data. Similarly to uniform data, the number of multicriteria function evaluations is a little lower and requires less time for the ESFS algorithm.

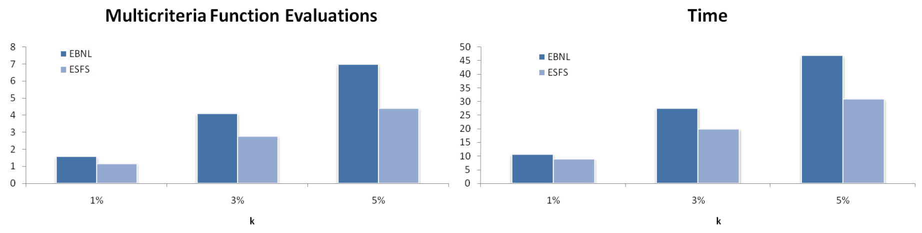

Fig. 2. Real data

7 Conclusions and Future Work

Top-k Skyline queries determine exactly the Top-k considering a score function from a partially ordered set stratified into subsets of non-dominated tuples in terms of a multicriteria function. Existing algorithms cannot solve Top-k Skyline problems. On one hand, Skyline algorithms do not restrict the result to exactly k tuples. On the other hand, Top-k algorithms might return results that would be false under Skyline criteria. Therefore, we have extended two algorithms for evaluating Top-k Skyline queries in relational databases. Both algorithms were integrated into PostgreSQL in the quest for better performance. Both algorithms are complete and build less strata than a naive solution. Initial experimental results show that ESFS performs less multicriteria function evaluations and requires less evaluation time than EBNL.

PostgreSQL does not optimize Top-k Skyline queries, so it does not benefit from full integration with the relational engine and fails to take advantage of the interesting orders that could be present in the data. In the future, we plan to extend PostgreSQL optimizer so that it considers Top-k Skyline queries when calculating costs and planning the execution.

Also, EBNL and ESFS could be more efficient if they would not require one table scan per stratum, this should be discussed in the future. Finally, an asymptotic complexity analysis should be performed to compare the proposed algorithms with the straightforward implementation of Top-k skyline.

Acknowledgments

This work was supported by FONACIT under Project G-2005000278.

References

1. Balke, W-T., Güntzer, U., Kiebling, W.: Towards Efficient Multi-Feature Queries in Heterogeneous Environments. In: Proceedings of the IEEE International Conference on Information Technology: Coding and Computing (ITCC), pp. 622–628 (April 2001)
2. Balke, W-T., Güntzer, U., Zheng, J.: Efficient Distributed Skylining for Web Information Systems. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 256–273. Springer, Heidelberg (2004)
3. Balke, W-T., Güntzer, U.: Multi-Objective Query Processing for Database Systems. In: Proceedings of the International Conference on Very Large Databases (VLDB), pp. 936–947 (September 2004)
4. Bruno, N., Gravano, L., Marian, A.: Evaluating Top-k Queries over Web-Accessible Databases. In: Proceedings of International Conference on Data Engineering (ICDE), vol. 29(4), pp. 319–362 (2002)
5. Carey, M., Kossman, D.: On saying Enough Already! in SQL. In: Proceedings of the ACM SIGMOD Conference on Management of Data, pp. 219–230 (May 1997)
6. Carey, M., Kossman, D.: Reducing the Braking Distance of a SQL Query Engine. In: Proceedings of VLDB, pp. 158–169 (August 1998)
7. Chang, K., Hwang, S-W.: Optimizing Access Cost for Top-k Queries over Web Sources: A Unified Cost-Based Approach. Technical Report UIUCDS-R-2003-2324, University of Illinois at Urbana-Champaign (March 2003)
8. Fagin, R.: Combining Fuzzy Information from Multiple Systems. Journal of Computer and System Sciences (JCSS) 58(1), 216–226 (1996)
9. Godfrey, P., Shipley, R., Gryz, J.: Maximal Vector Computation in Large Data Sets. In: Proceedings of VLDB, pp. 229–240 (2005)
10. Goncalves, M., Vidal, M.E.: Preferred Skyline: A Hybrid Approach Between SQLf and Skyline. In: Andersen, K.V., Debenham, J., Wagner, R. (eds.) DEXA 2005. LNCS, vol. 3588, pp. 375–384. Springer, Heidelberg (2005)
11. Goncalves, M., Vidal, M.E.: Top-k Skyline: A Unified Approach. In: Proceedings of OTM (On the Move) 2005 PhD Symposium, pp. 790–799 (2005)
12. Huang, Z., Jensen, C.S., Lu, H., Ooi, B.C.: Skyline Queries Against Mobile Lightweight Devices in MANETs. In: Proceedings of ICDE, pp. 66–77 (2006)
13. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K.: Supporting Top-k Join Queries in Relational Databases. In: Proceedings of VLDB, pp. 754–765 (2003)
14. Kossman, D., Ransak, F., Rost, S.: Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In: Proceedings of VLDB, pp. 275–286 (2002)
15. Lo, E., Yip, K., Lin, K-I., Cheung, D.: Progressive Skylining over Web-Accessible Databases. Journal of Data and Knowledge Engineering 57(2), 122–147 (2006)
16. Natsev, A., Chang, Y-CH., Smith, J.R., Li, CH.-S., Vitter, J.S.: Supporting Incremental Join Queries on Ranked Inputs. In: Proceedings of VLDB, pp. 281–290 (2001)
17. Nepal, S., Ramakrishnan, M.V.: Query Processing Issues in Image (Multimedia) Databases. In: Proceedings of ICDE, pp. 22–29 (1999)
18. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive Skyline Computation in Database Systems. ACM Transactions Database Systems 30(1), 41–82 (2005)
19. Tan, K-L., Eng, P-K., Ooi, B.C.: Efficient Progressive Skyline Computation. In: Proceedings of VLDB, pp. 301–310 (2001)
20. Zagat Survey Guides: available at `http://www.zagat.com`