The Multi-Relational Skyline Operator

Wen Jin[1] Martin Ester[1] Zengjian Hu[1] Jiawei Han[2]
[1]School of Computing Science
Simon Fraser University, Canada
{wjin,ester,zhu}@cs.sfu.ca
[2]Department of Computer Science
University of Illinois at Urbana-Champagne, USA
hanj@cs.uiuc.edu

Abstract

Most of the existing work on skyline query has been extensively used in decision support, recommending systems etc, and mainly focuses on the efficiency issue for a single table. However the data retrieved by users for the targeting skylines may often be stored in multiple tables, thus require to perform join operations among tables. As a result, the cost on computing skylines on the joined table will be increased dramatically due to its potentially increasing cardinality and dimensionality. In this paper, we systematically study the skyline operator on multi-relational databases, and propose solutions aiming to seamlessly integrating state-of-the-art join methods into skyline computation. Our experiments not only demonstrate that the proposed methods are efficient, but also show the promising applicability of extending skyline operator to other typical database operators such as join and aggregates.

1 Introduction

The skyline operator [1] returns those objects not dominated by any other objects, where the *"dominate"* relationship between object p and q is defined as follows: if the value of p on each dimension is not larger than that of q and strictly smaller on at least one dimension, then p dominates q (denoted as $p \succ q$). Most of the existing work on skyline queries for databases mainly discusses the computation efficiency in one single relation table. While in many database applications, users are often interested in querying skylines from multiple relational tables which require join operations. Formally, we call the skyline operator over a multi-relational joined table $A_1 \bowtie A_2 \ldots \bowtie A_k$ as the *multi-relational skyline operator*, or simply as the *skyline join operator*[1], denoted by $A_1 \bowtie_s A_2 \ldots \bowtie_s A_k$ where $A_1, \ldots, A_k$ are relations.

[1]In this paper, terms of "multi-relational skyline operator", "skyline join operator" or "joined skylines" all refer to the skyline objects in the joined table

The following example skyline queries aim to find model customers in TPC-D datasets (www.tpc.org) for the company such that the higher the *account balance*, the lower the *age* of a customer (in relation *customer*), and also the higher the *quantity* of parts and the *amount* of price (in relation *order*) this customer orders, the "better" this customer.

Example 1 *Given customer table* Customer (**<u>CNum</u>**, Age, Account Balance) *(as shown in Table 1) and part order table* Order (**<u>ONum</u>**, CNum, PNum, Quantity, Amount) *(as shown in Table 2). Consider the following queries:*
Question (1):*Who are those young customers with high account balance and having* **ONE** *order with high quantities of parts and high amount of price?*
Question (2):*Who are those young customers with high account balance and high quantities of parts order with high* **TOTAL** *amount of price (over all orders of this customer)?*

Answer: *The SQL statements for answering the above questions are as follows:*

SELECT *
FROM *Customer C, Order O*
WHERE $C.CNum = O.CNum$
SKYLINE *OF* $C.Age$ **Min**, $C.AccountBalance$ **Max**, $O.Quantity$ **Max**, $O.Amount$ **Max**;

SELECT $C.CNum$, **SUM**$(O.Quantity)$, **SUM**$(O.Amount)$
FROM *Customer C, Order O*
WHERE $C.CNum = O.CNum$
SKYLINE *OF* $C.Age$ **Min**, $C.AccountBalance$ **Max**, $O.Quantity$ **Max**, $O.Amount$ **Max**
GROUP By $C.CNum$;

The join results and the skyline objects in $Customer \bowtie Order$ *where (1) attributes* $CNum$ *of* C *and* $CNum$ *of* O *are* join attributes*; (2) attributes* Age, $AccountBalance$, $Quantity$ *and* $Amount$ *are* descriptive attributes *participating in skyline evaluation (under shaded regions), are shown in Table 3 and Table 4 respectively (***skyline objects are in bold font***)*[2]*. The joined skylines may contain tuples*

[2]For simplicity, we denote $C.CNum$ as variables D_1, Age as D_2,

CNum	*Age*	*Account Balance* ($)
101	**35**	**90k**
102	40	40k
103	50	78k
104	**35**	**90k**
105	58	90k

Table 1. *Customer(C)* **Table**

ONum	**CNum**	PNum	*Quantity*	*Amount*($)
1	101	001	1	274
2	101	001	6	1644
3	**102**	**002**	**10**	**1999.9**
4	103	003	1	400
5	104	004	5	900
6	104	004	6	1080
7	105	005	2	1900

Table 2. *Order(O)* **Table**

that are not in the skyline of the individual input tables, such as the tuple with $D1 = 105$*. Notice that the skyline queries w.r.t. (1) and (2) return different answers due to the effect of the aggregate constraints being pushed on descriptive attributes. A customer may be a best buyer according to the quantity and price for one sale, but may not be a best buyer if considering all the orders he/she made.* □

D1	*D2*	*D3*	D4	**D5**	*D6*	*D7*
101	35	90k	1	101	1	274
101	**35**	**90k**	**2**	**101**	**6**	**1644**
102	**40**	**40k**	**3**	**102**	**10**	**1999.9**
103	50	78k	4	103	1	400
104	35	90k	5	104	5	900
104	35	90k	6	104	6	1080
105	**58**	**90k**	**7**	**105**	**2**	**1900**

Table 3. Joined Table of *Customer* **and** *Order* **(1)**

The join operation in Example 1 is *non-reductive* [3] (this assumption holds in the remaining of the paper) in terms that the size of joined table is larger or equal to that of any table participating in the join operation. Also the tuples have higher dimensions, so that the cost of applying skyline operator after joins would be much more expensive[3]. For arbitrary join operations, since both the cardinality and dimensionality of the joined table might increase, so the cost

$AccountBalance$ as D_3, $ONum$ as D_4, $O.CNum$ as D_5, $Quantity$ as D_6 and $Amount$ as D_7 (for simplicity here we omit $PNum$ in the joined table), and these notations are used in the remaining of this paper.

[3]The cost of skyline computation methods increases w.r.t. the increase of dimensionality.

D1	*D2*	*D3*	**D5**	*SUM(D6)*	*SUM(D7)*
101	35	90k	101	7	1918
102	**40**	**40k**	**102**	**10**	**1999.9**
103	50	78k	103	1	400
104	**35**	**90k**	**104**	**11**	**1980**
105	58	90k	105	2	1900

Table 4. Joined Table of *Customer* **and** *Order* **(2)**

of finding skylines in the joined table will be even larger. How to develop efficient methods to share the join processing with skyline computation is central to the skyline query optimization on multiple relations. Motivated by the above observations, in this paper, we systematically study the multi-relational skyline operator and make the following contributions: **(1)** We study the problem of skyline computation with or without aggregate constraints in multiple tables when join operation works on them. **(2)** We propose different approaches which aim to combine state-of-the-art join methods into skyline computation for any single join operation and extend to the case of multiple join operations. **(3)** Our experiments on TPC-D benchmark demonstrate the efficiency and scalability of our proposed methods.

The rest of the paper is organized as follows. Section 2 introduces the background and preliminaries, and Section 3 introduces the approaches of answering skylines over a joined table. We present experimental results in section 4 and conclude the paper in section 5.

2 Background and Preliminaries

After Borzsonyi et al. introduced the skyline operator [1] in database community, a set of algorithms were developed for variants of this problem. Two disk-resident algorithms: block-nested loops (BNL) method and an improved divide-and-conquer method were proposed in [1]. Chomicki et al. proposed a sort-first-skyline (SFS) method [2] and Godfrey et al. then proposed a skyline algorithm LESS based on SFS, which achieves a better average performance [4]. In [9], the authors proposed a bitmap index method and a "minimum value list" method (indexed by a specialized B-tree) to find skylines. Kossmann et al. present an nearest neighbor search-based method using R-tree [5]. Papadias et al. proposed a progressive method to find skyline with optimal number of node accesses [7] with R-tree indexes. Recent study on skyline includes subspace skyline computation [6], [12], [10] and [11].

Let denote a relation X in an n-dimensional space $D = (D_1, \ldots, D_n)$, where dimensions $D_1, \ldots, D_n$ are in the domain of numbers. For any $p \succ q$, q is called a dominated object, and denote the set of objects dominating q as $Dom(q)$.

Given two relations $A(d_1, \ldots, d_i, d_{i+1}, \ldots, d_{n_1})$ and $B(d'_1, \ldots, d'_j, d'_{j+1}, \ldots, d'_{n_2})$, where attributes $d_1, \ldots, d_i$ in A and $d'_1, \ldots, d'_j$ in B are called **join attributes** such that they are only used in the join operation[4]. Attributes $d_{i+1}, \ldots, d_{n_1}$ in A and attributes $d'_{j+1}, \ldots, d'_{n_2}$ in B are called **descriptive attributes** that participate in the evaluation of skylines. Corresponding to the scenarios in Example 1, we generalize two typical skyline problems as follows.

Problem 1 (Skyline Join Problem) *Given two relations $A(d_1, \ldots, d_i, d_{i+1}, \ldots, d_{n_1})$ and $B(d'_1, \ldots, d'_j, d'_{j+1}, \ldots, d'_{n_2})$, find skylines over the joined table $A \bowtie B$ excluding the join attributes.*

Problem 2 (Skyline Join with Aggregate Constraint Problem) *Given two relations $A(d_1, \ldots, d_i, d_{i+1}, \ldots, d_{n_1})$ and $B(d'_1, \ldots, d'_j, d'_{j+1}, \ldots, d'_{n_2})$, find skylines over the joined table $A \bowtie B$ where aggregate constraints on descriptive attributes in B are applied on corresponding tuples in A. Here the aggregates include Max, Min, COUNT, SUM and AVG etc.*

In each table, we can group tuples according to the values of *join attributes* in ascending order. Every tuple belongs to one of three cases: (1) $\mathbf{LS(S)}$ means this tuple is a *local skyline* in its group and also a *skyline in the whole table*. (2) Label $\mathbf{LS(N)}$ means this tuple is only a *local skyline* in the group but *not a skyline in the whole table*. (3) Label $\mathbf{LN(N)}$ means this tuple is *locally not a skyline* in the group (and of course it is *not a skyline in the whole table*). If we use symbol "$\oplus$" to denote the concatenate operator to combine two joinable tuples in A and B into a joined tuple, then each joined tuple in the Table $A \bowtie B$ has the following six cases: $LS(S) \oplus LS(S)$, $LS(S) \oplus LS(N)$, $LS(S) \oplus LN(N)$, $LS(N) \oplus LN(N)$, $LN(N) \oplus LN(N)$ and $LS(N) \oplus LS(N)$. We have the following properties:

Lemma 2.1 *The joined tuple $LS(S) \oplus LS(S)$ or $LS(S) \oplus LS(N)$ is also a skyline in $A \bowtie B$. The joined tuple $LS(S) \oplus LN(N)$, $LS(N) \oplus LN(N)$, $LN(N) \oplus LN(N)$ cannot be a skyline in $A \bowtie B$.*

Property 2.2 *The joined tuple $LS(N) \oplus LS(N)$ may be or may not be a skyline in $A \bowtie B$.*

In this paper, we focus on the skyline computation over $A \bowtie B$ where a one-to-many relationship [8] exists between A and B such that the join operation is performed if A has a primary key matching a foreign key in B. The case of the join operation over A and B with many-to-many relationship [8] can be solved by introducing a third table C including attributes of B and join attribute of A, then perform join between A and C, and join between the join results and B.

[4]It is often the case that join attributes refer to the primary key in A and foreign key(s) in B. However, in general case a join attribute could be any attribute. For ease of analysis, we assume whenever an attribute participates in the join, it does not participate in skyline evaluation.

3 Algorithms

We present in this section different approaches of integrating the skyline computation into state-of-the-art join algorithms. Denote the boolean function of evaluating join condition between $p \in A$, $q \in B$ as $\theta(p, q)$.

3.1 A Naive Approach for Skyline Join

A naive approach for the skyline join operator works as follows: we first compute the join, then apply existing skyline algorithms on the joined relation to find the corresponding skyline objects.

As the join operation is assumed to be *non-reductive* [3], the number of attributes of the joined table is larger than that of each single table participating in join operation, so the cost of running skyline algorithm in the entire joined table is much more expensive than in any single table. If we consider the common case of the increase in both cardinality and dimensionality of the joined table, the cost of the naive approach would be even more expensive.

3.2 Integrating with Sort-Merge Join

Basic Approach As illustrated in the previous section, we can find the joined skyline only based on the information of being skyline or being dominated for tuple $p \in A$ and $q \in B$ during the join processing. The only uncertainty case that is as described in Property 2.2 when two tuples participating in the join are both labeled as "LS(N)", can be solved by the following lemma.

Lemma 3.1 *Let $p \in A$ and $q \in B$ be joinable tuples labeled as "LS(N)", if $\theta(p', q') = false$ for all $p' \in Dom(p), q' \in Dom(q)$, then $p \oplus q$ is a joined skyline object; otherwise $p \oplus q$ cannot be a joined skyline object.*

Thus, the computation of joined skylines can be performed by integrating with the most efficient sort-merge join algorithm [8]. Essentially, during the sort-merge join processing, if two join tuples are matching in their join attributes, according to the label of "$LS(S)$", "$LS(N)$" or "$LN(N)$" for tuple $p \in A$ and $q \in B$, we decide whether the joined tuple $p \oplus q$ is a skyline object in $A \bowtie B$ depending on whether they satisfy the properties illustrated in Lemmas 2.1 and 3.1.

This method can also be used to solve Problem 2 which includes some aggregation operator with a slightly modified algorithm by first treating the tuples in the same group as a single tuple with the aggregated values, then applying the same procedure to identify skylines.

Efficient Dominators Match Using R-trees To overcome the problem of computing the dominators $Dom(p)/Dom(q)$ for each tuple p with label "LS(N)"

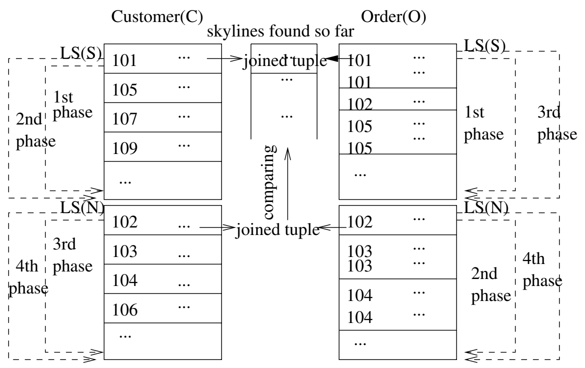


Figure 1. Comparing $a_i \oplus b_j$ with skylines found so far in joined table

which incurs non-trivial cost when the dataset is large, we propose to use R-tree to facilitate the search of dominators. Basically, the descriptive attributes of each tuple in each table are indexed in R-tree (as a typical multi-dimensional index structure, R-tree has been widely used). The dominators of any object can be easily obtained by range queries in R-tree similar to the work in [7], then the dominators are sorted with respect to values of the join attribute, and the match of dominators is tested. To facilitate the match, we can record the range of join attribute in each MBR node, and prune the unnecessary comparisons as much as possible where join attribute value range does not overlap.

Comparing with Joined Skylines During Join Process To eliminate the need for dominating test, we propose a method for the computation of joined skylines based on comparing each $LS(N) \oplus LS(N)$ tuple efficiently with the joined skylines found so far in the join process. Here we still follow the strategy of sort-merge join, but adopt two modifications: (1) we discard tuples with label "LN(N)" since they have no contribution to the final skylines in the joined table. (2) we sort each table as follows: (a) the primary sorting for each table is by the order of $LS(S) < LS(N)$, so tuples with label "LS(S)" are placed before tuples with label "LS(N)" in each table, as shown in Figure 1; (b) within each group of "LS(S)" and "LS(N)" obtained by (a), the secondary sorting is performed by the join attribute value of each tuple. The reason of these modifications is that we can determine which tuples in the joined table are skylines as early as possible, and also can reduce the sizes of input tables as much as possible. Now the sort-merge join consists of four phases. The first three phases are $LS(S) \oplus LS(S)$, $LS(S) \oplus LS(N)$ and $LS(N) \oplus LS(S)$, are indicated by those dashed lines in Figure 1.

The above tuples which can be simply combined if matching up, are skylines S (labeled as "LS(S)") in the joined table, and they are sorted with entropy value in ascending order for checking dominating relationship later. The fourth phase is $LS(N) \oplus LS(N)$ (the dashed lines with "4th" in Figure 1). If any "LS(N)" matches "LS(N)", compare with the joined tuple in S, if dominated, it cannot be a joined skyline; otherwise save to the temporary set S'. Later on, any new $LS(N) \oplus LS(N)$ will compare with S and S'.

3.3 Integrating with Nested-Loop Join

We can also find joined skyline by integrating with the nested-loop join [8] with the intuition that if the joined tuples obtained during the join process keep an "order" in some *descriptive attributes*, then if a new joined tuple is not dominated by joined skylines found so far, it is easily identified as a joined skyline. Here we need to sort each join table in one of *descriptive attributes*, and meanwhile maintain the information about different labels as described previously. Note that, those labels are obtained in the Group-By operation with respect to *join attributes* in each table.

4 Experimental Evaluation

In this section, we report the results of our experimental evaluation. All methods proposed in this paper were implemented using Microsoft Visual C++ V6.0, including (1) the improved sort-merge join based skyline methods by (i) using R-tree MBRs (noted as **SMJS1**), and by (ii) comparing with joined skylines during the join process (noted as **SMJS2**); (2) the block nested-loop join based skyline method (noted as **NLJS**); and (3) the naive-based skyline algorithm (noted as **Naive**). Using the data generator provided by www.tpc.org, we generated several types of TPC-D benchmark tables: *Customer*(each tuple has 44 bytes), *Order*((each tuple has 84 bytes)) and *Part* (each tuple has 60 bytes) [5] . For each type of table, we generated data sets with different sizes (from $10,000$ to $1,000,000$ tuples).

Dataset1	C, 20k	O, 100x1k	P, 20x1k
Dataset2	C, 50k	O, 200x1k	P, 50x1k
Dataset3	C, 100k	O, 500x1k	P, 100x1k
Dataset4	C, 200k	O, 1000x1k	P, 200x1k

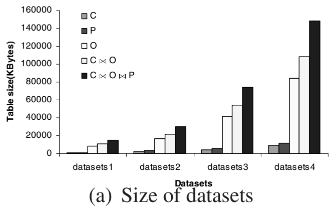
Table 5. Cardinality of different datasets

By choosing four different datasets as shown in Table 5, the physical sizes and the number of skylines of *C*, *O* *P*, *C*⋈*O* and *C*⋈*O*⋈*P* are listed in Figure 2(a) and Figure 2(b)

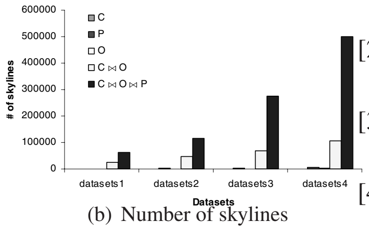
[5]Customer relation has 3 descriptive attributes: account balance, salary, age; Order relation has 7 descriptive attributes: quantity, total price, priority, discount, ship cost, tax, delivery duration; Part relation has 5 descriptive attributes:quantity, price, cost, discount range, weight, size. In the legend of figures, C-Customer, O-Order, P-Part, $CjoinO$-C ⋈ O, C join O join P-C ⋈ O ⋈ P

respectively, with respect to different cardinalities. In both cases, the size differences are evident and become larger with more cardinalities and more participating relations.

The computation of joined skylines with aggregate constraints uses less time due to the smaller size of input tables after the aggregation. The run time comparisons of different methods for computing joined skylines with/without aggregate constraints with respect to different sizes of the joined table $C \bowtie O$ (with totally 10 descriptive attributes) are depicted in Figure 3(a) and Figure 3(b) respectively. Here, we typically test the "SUM" aggregate as mentioned in Example 1. In both cases, our proposed methods run much faster than **Naive** method. In particular, **SMJS2** finishes first, and **SMJS1** takes less time than **NLJS**. The relationship between run time and the dimensionality of the joined table are illustrated in Figure 4(a) and Figure 4(b) respectively. The joined table is obtained from the joining of O of 500,000 records and P of 100,000 records. Each time we choose 2, 3, 4 and 5 dimensions per table respectively to participate join operation, thus the joined table has the dimensionality of 4, 6, 8 and 10 respectively. It shows the same ranks of run time as the test of run time w.r.t. different sizes of joined tables.

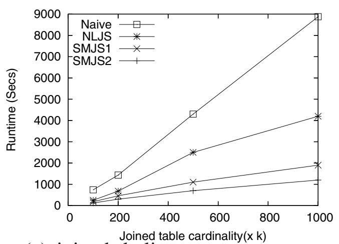


(a) Size of datasets

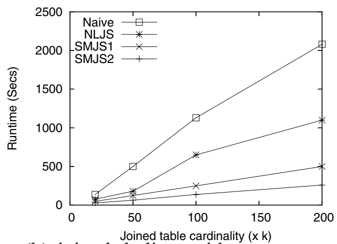


(b) Number of skylines

Figure 2. The size of datasets and #of skylines



(a) joined skylines w.o. aggregates

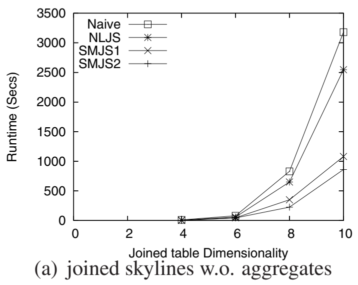


(b) joined skylines with aggregates

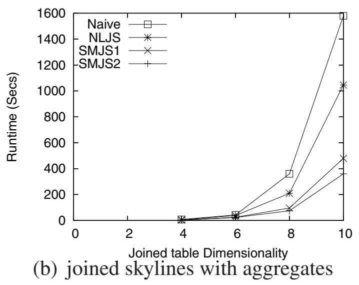
Figure 3. Runtime vs cardinality

5 Conclusion

In this paper, we introduce the skyline operator over multi-relational tables, and propose efficient algorithms by incorporating state-of-the-art join methods into skyline computation. The experiments on TPC-D datasets demonstrate the efficiency and scalability of the proposed methods. We believe that this research does not only meaningfully extend the skyline operator to the multi-relational database systems, but also indicate the interesting topics such as joined skylines in the case of updated data and other types of aggregates.



(a) joined skylines w.o. aggregates



(b) joined skylines with aggregates

Figure 4. Runtime vs dimensions

References

[1] S. Borzsonyi, D. Kossmann and K. Stocker. The Skyline Operator. In *ICDE*, 2001.

[2] J. Chomicki, P. Godfrey J. Gryz and D. Liang. Skyline with Pre-sorting. In *ICDE*, 2003.

[3] M. Carey and D.Kossmann. On Saying "Enough Already!" in SQL. In *SIGMOD*, 1997.

[4] P. Godfrey, R. Shipley and J. Gryz. Maximal Vector Computation in Large Data Sets. In *VLDB*, 2005.

[5] D. Kossmann , F. Ramsak and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. *VLDB*, 2002.

[6] J. Pei, W. Jin, M. Ester and Y. Tao. Catching Best Views of Skyline:A Semantic Approach Based on Decisive Subspaces. *VLDB*, 2005.

[7] D. Papadias, Y. F. Tao, G. Fu and B. Seeger. An Optimal and Progressive Algorithm for Skyline Queries. In *SIGMOD*, 2003.

[8] R. Ramakrishnan and J. Gehrke. Database Management Systems 3rd Edition, McGraw-Hill 2004.

[9] K. Tan, P. Eng and B.Ooi. Efficient Progressive Skyline Computation. In *VLDB*, 2001.

[10] Y. F. Tao, X. K. Xiao and J. Pei. SUBSKY: Efficient Computation of Skyline in Subspace. In *ICDE*, 2006.

[11] T. Xia and D. Zhang. Refreshing the Sky:The Compressed Skycube with Efficient Support for Frequent Updates. In *SIGMOD*, 2006.

[12] Y. Yuan, X. Lin, Q. Liu and W. Wang, J.X. Yu and Q. Zhang. Efficient Computation of the Skyline Cube. In *VLDB*, 2005.