SkySuite: A Framework of Skyline-Join Operators for Static and Stream Environments*

Mithila Nagendra
School of Computing, Informatics,
and Decision Systems Engineering
Arizona State University, Tempe, AZ, USA
mnagendra@asu.edu

K. Selçuk Candan
School of Computing, Informatics,
and Decision Systems Engineering
Arizona State University, Tempe, AZ, USA
candan@asu.edu

ABSTRACT

Efficient processing of skyline queries has been an area of growing interest over both static and stream environments. Most existing static and streaming techniques assume that the skyline query is applied to a single data source. Unfortunately, this is not true in many applications in which, due to the complexity of the schema, the skyline query may involve attributes belonging to multiple data sources. Recently, in the context of static environments, various hybrid *skyline-join* algorithms have been proposed. However, these algorithms suffer from several drawbacks: they often need to scan the data sources exhaustively in order to obtain the set of skyline-join results; moreover, the pruning techniques employed to eliminate the tuples are largely based on expensive pairwise tuple-to-tuple comparisons. On the other hand, most existing streaming methods focus on single stream skyline analysis, thus rendering these techniques unsuitable for applications that require a real-time "join" operation to be carried out before the skyline query can be answered. Based on these observations, we introduce and propose to demonstrate `SkySuite`: a framework of *skyline-join* operators that can be leveraged to efficiently process skyline-join queries over both static and stream environments. Among others, `SkySuite` includes (1) a novel Skyline-Sensitive Join (`SSJ`) operator that effectively processes skyline-join queries in static environments, and (2) a Layered Skyline-window-Join (`LSJ`) operator that incrementally maintains skyline-join results over stream environments.

1. INTRODUCTION

Recently, there has been a growing interest in the efficient processing of skyline queries over both static [5, 2] and stream environments [7, 14]. Given a set, D, of data points in a feature space, the *skyline* of D consists of the points that are not dominated[1] by any other data point in D [2]. Intuitively, the skyline is a set of *interesting* points that help paint the "bigger picture" of the data in question, providing insight into the diversity of the data across different features.

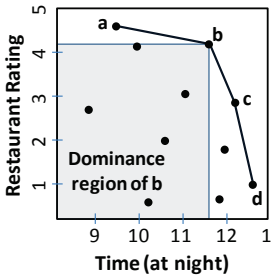


Figure 1: Skyline of late-night restaurants

Searching for non-dominated data is valuable in many applications that involve multi-criteria decision making [11]. For instance, students in a university who stay up late at night and need a snack at odd hours might find the skyline of late-night restaurants useful. Figure 1 shows the ratings and closing times of a set of restaurants: the points that are connected represent restaurants that are part of the skyline; this includes highest-rated restaurants that are open late into the night. Other restaurants are not part of the skyline because they are dominated in terms of time and/or rating by at least one restaurant that is in the skyline. The shaded area in Figure 1 is the dominance region of restaurant b: for any restaurant in this range, b is either open till a later time and/or has a better rating; therefore b is said to be more *interesting* than all restaurants it dominates.

A particular shortcoming of existing static and stream skyline algorithms is that they primarily focus on single-source skyline processing in which all required skyline attributes are present in the same source. However, there are many applications in both static and stream environments that require integration of data from different sources. In such scenarios, the skyline query may involve attributes belonging to different data sources, thus making the join operation an integral part of the overall process. For instance, in static environments integrated *skyline-join* queries maybe necessary over complex schemas in which the data is distributed onto many sources, whereas in stream environments such integration is needed for streams that originate from

This work is supported by an NSF grant (#1116394 – RanKloud: Data Partitioning and Resource Allocation Strategies for Scalable Multimedia and Social Media Analysis*) and a KRF grant (*A Framework for Real-time Context Monitoring in Sensor-rich Personal Mobile Environments*).

Proceedings of the VLDB Endowment, Vol. 6, No. 12

[1]A point dominates another point if it is as good or better in all dimensions, and better in at least one dimension.

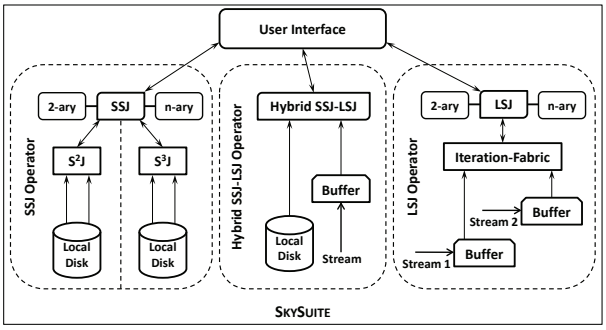


Figure 2: The SkySuite framework

different sensors or from multiple sources in a distributed publish/subscribe architecture.

Going back to our earlier example, in addition to the time and restaurant rating attributes shown in Figure 1, students might also consider the distance of a restaurant to the university to be a factor in their decision-making process. If this information is available from a different source, we would then need to join the relevant sources in order to obtain the restaurants that are part of the skyline.

Motivated by the above observations, we propose SkySuite: a framework of *skyline-join* operators that can be used to process skyline-join queries over both static and stream environments (Figure 2). In particular, we demonstrate (1) the Skyline-Sensitive Join (SSJ) operator [8] that processes skyline-join queries in static environments, and (2) the Layered Skyline-window-Join (LSJ) operator [9] that incrementally maintains skyline-joins in stream environments.

The rest of the paper is structured as follows: in Section 2, we give an overview of the existing work in the field of skyline query processing. Section 3 presents the suite of skyline-join operators. In Section 4, we discuss the demonstration scenarios. Lastly, we conclude the paper in Section 5.

2. RELATED WORK

The task of finding the non-dominated set of data points was attempted by Kung *et al.* [5] in 1975 under the name of the maximum vector problem. Kung's algorithm lead to the development of various skyline algorithms designed for static [2, 16] and stream environments [7, 14].

2.1 Skylines over a Single Static Data Source

Borzsonyi *et al.* [2] were the first to coin and investigate the *skyline* computation problem in the context of databases. Later contributions to skyline query processing include sort-based techniques (*SFS* [3]), progressive methods (*bitmap* and *index* [13]), and online algorithms [10].

2.2 Skylines on Multiple Static Data Sources

Some of the prior work on skylines over multiple static data sources include [4, 12, 16]. Sun *et al.* [12] introduce an operator called *skyline-join*, and two algorithms to support skyline-join queries. The first extends the *SaLSa* algorithm [1] to cope with multiple relations, whereas the second algorithm (*Iterative*) prunes the search space iteratively.

More recently, Vlachou *et al.* [16] introduced the *Sort-First-Skyline-Join (SFSJ)* algorithm that fuses the identification of skyline tuples with the computation of the join. SFSJ provides a way to prune the input tuples if they do not contribute to the set of skyline-join results, thus reducing the number of generated join results and dominance checks.

However, SFSJ does not carry out the pruning in a block-based manner and largely depends on time-consuming tuple-to-tuple comparisons to find the pruned region. The Skyline-Sensitive Join (SSJ) operator, demonstrated in this paper, overcomes this drawback by pruning the join space in terms of blocks of data, as opposed to individual data points, thereby avoiding excessive point-to-point dominance checks.

Over the last decade, the advent of a wide array of stream-based applications has necessitated a push towards the development of algorithms that take into consideration the constant changes in stream environments. The following sections provide an overview of the existing work in the fields of skyline and join query processing over streaming data.

2.3 Join Processing over Data Streams

[17] presents a symmetric hash join method that is optimized for in-memory performance. Following this, a plethora of techniques have been developed for processing join queries over data streams [6, 15]. Many of these focus on eliminating redundancy in join processing to maximize the output rate [6]. Others focus on memory; they present join processing and load shedding techniques that minimize loss in accuracy when the memory is insufficient [15].

2.4 Skyline Processing over Data Streams

As mentioned earlier, in the conventional setting of static data, there is a large body of work for both single-source skyline processing [2, 3, 1] and multiple source skyline-join processing [12, 16]. These methods assume that the data is unchanging during query execution and focus on computing a single skyline rather than continuously tracking skyline changes. Recently, several algorithms have been developed to track skyline changes over data streams. These methods continuously monitor the changes in the skyline according to the arrival of new tuples and expiration of old ones.

Data stream skyline processing under the sliding window model is addressed in [7] and [14]. An important issue that needs to be addressed here is the expiration of skyline objects. To tackle this issue, Tao *et al.* present the *Eager* algorithm [14] that employs an event list, while Lin *et al.* propose a method (*StabSky*) that leverages *dominance graphs* [7]. Both these methods memorize the relationship between a current skyline object and its successor(s). Once skyline objects expire, their successor(s) can be presented as the updated skyline without any added computation.

The above-mentioned approaches focus on skyline queries in which the skyline attributes belong to a single stream, thus rendering them inapplicable to the problem of computing skyline-joins over multiple streams. In this paper, we demonstrate the novel Layered Skyline-window-Join (LSJ) operator; this operator is first of its kind for answering *skyline-window-join* (SWJ) queries over data streams.

3. SKYSUITE

This section introduces SkySuite: a framework of *skyline-join* operators for processing skyline-join queries over both static and stream environments (Figure 2). In particular, we explain the methodologies behind the Skyline-Sensitive Join (SSJ) and Layered Skyline-window-Join (LSJ) operators.

3.1 SSJ Operator for Static Environments

At the core of the SSJ operator are two skyline-join algorithms, namely S^2J (*skyline-sensitive join*) and S^3J (*symmetric skyline-sensitive join*) [8]. Both S^2J and S^3J are single-

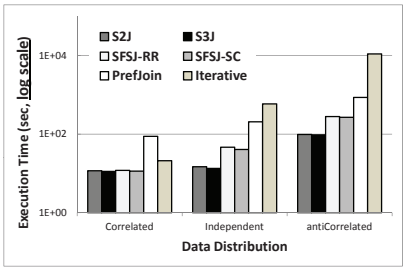
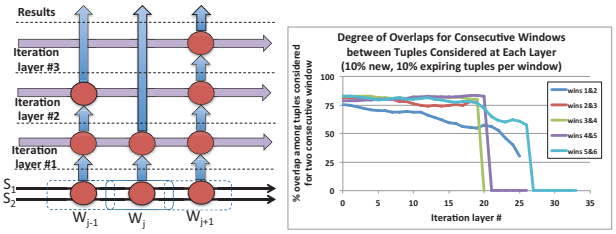


Figure 3: S^2J and S^3J efficiently process skyline-join queries over static data [8]



(a) (b)

Figure 4: (a) Viewing layers as separate "virtual streams" that feed the upper layers of iteration; (b) Sample SWJ execution for 6 consecutive windows (10% new and 10% expiring tuples per window): the plots show that the skyline-join process iterate somewhere between 20 to 35 times for different windows and the overlaps (among consecutive windows) of tuples considered at different layers of iterations remain high across layers of iteration

pass, two-way skyline-join algorithms that avoid tuple-to-tuple dominance checks wherever possible. These algorithms rely on a novel *layer/region pruning* (*LR-pruning*) strategy in order to avoid excessive pairwise dominance checks.

The key features of S^2J are as follows:

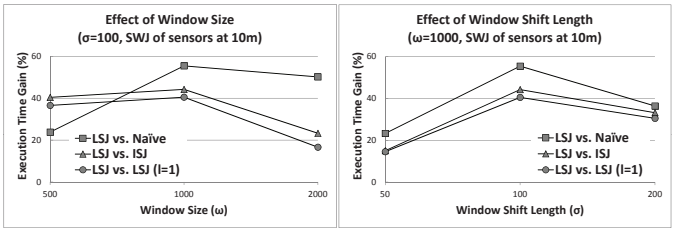
- The tuples in the outer table are organized into *layers* of dominance.
- The tuples of the inner table are clustered into *regions* based on the Z-values of the skyline attributes to support block-based pruning.
- A *trie*-based data structure on the inner table keeps track of the so-called *dominated*, *not-dominated*, and *partially-dominated* regions of the inner table *relative to the layers of the outer table.*
- S^2J obtains the skyline set by scanning the outer table, only once, while pruning the inner table.

S^3J is similar to S^2J in principle, but repeatedly swaps the roles of the outer and inner tables. One key outcome of this strategy is that (unlike S^2J, where the outer table is fully scanned), S^3J rarely needs to scan any of the input tables entirely in order to obtain the set of skyline points.

The effectiveness of S^2J and S^3J compared to the SFSJ methods [16], PrefJoin [4], and iterative skyline-join [12] can be seen in the sample experimental result shown in Figure 3. Please refer to [8] for further details.

3.2 LSJ Operator for Stream Environments

The LSJ operator processes SWJ queries over two data streams by maintaining skyline-join results in a layered, incremental manner. It continuously monitors the changes in the data streams, and eliminates redundant work between consecutive windows by leveraging shared skyline objects across all iteration layers of skyline-join processing.



(a) Effect of window size (b) Effect of shift length

Figure 5: The LSJ operator effectively handles skyline-join queries over streaming data [9]

LSJ is based on the observation that the consecutive iterations of the algorithm, spanning multiple windows, can be viewed as separate *iteration layers* (Figure 4(a)). The key insight here is that *overlaps exist not only at the lowest data layer (across consecutive data windows), but also at the individual iteration layers, where the tuples processed can be considered as "virtual streams" that evolve from one window to the next* (see Figure 4(b) for a sample execution).

Therefore, we argue that if we naively execute the SWJ operation by applying the iterative skyline join algorithm separately for each window, we can end up with significant amount of redundant work. We further argue that if we can quickly identify and eliminate these per-layer overlaps, we can achieve significant savings in processing time.

Based on these insights, we develop the *iteration-fabric* [9]; this forms the backbone of the LSJ operator. The *iteration-fabric* helps combine the advantages of two existing skyline methods, *StabSky* [7] and *Iterative* [12], in developing a Layered Skyline-window-Join (LSJ) operator that maintains skyline-join results in an incremental manner by continuously monitoring the changes in the input streams and leveraging any overlaps that exist between the data considered at individual layers of consecutive sliding windows.

The efficiency of the LSJ operator compared to Naive, ISJ [9], and LSJ ($l = 1$) (where LSJ is applied only at the first layer of each window) is illustrated in the sample results shown in Figure 5. Please refer to [9] for further details.

4. USER INTERACTION SCENARIOS

This section describes the demonstration scenarios. We will use real data sets (JCI building energy simulation/observation[2], NBA[3] and Intel Berkeley Research[4]) and the TPC-H benchmark data sets[5]. Through an interactive graphical user interface, the attendees of this demonstration will be able to experience the suite of skyline-join operators up close and personal. Described next, are some example user interactive demonstration scenarios.

4.1 Two-way Skyline-Joins over Static Data

This scenario demonstrates how the SSJ operator is designed to handle skyline-joins between two static data sets.

[2]JCI is an energy IT company, with access to model, simulation, and sensory data for buildings of all types and sizes.
[3]http://skyline.dbai.tuwien.ac.at/datasets/nba/.
[4]http://db.csail.mit.edu/labdata/labdata.html.
[5]http://www.tpc.org/tpch/default.asp.

As seen in Figure 2, the SSJ operator utilizes the S^2J and S^3J algorithms, interchangeably, to execute skyline-join queries. This query scenario is executed over the NBA and TPC-H benchmark data sets. Attendees of this demonstration will be able to compare the performance of S^2J and S^3J against other algorithms, and will be able to observe the behaviour of the SSJ operator over different skyline-join queries.

EXAMPLE 1 (SSJ OPERATION). *Give two tables,* `Player-points (playerID, points, fieldGoals)` *and* `Player-assists (playerID, assists, freeThrows)`*, both derived from the NBA data set, a skyline-join query over* `Player-points` *and* `Player-assists` *could be:*

```
Skyline = SSJ * FROM Player-points P,
                Player-assists A,
                WHERE P.playerID = A.playerID,
                points MAX, fieldGoals MAX,
                assists MAX, freeThrows MAX.
```

This query equi-joins the tables on `playerID` *and returns results that are in the skyline based on the attributes* `points`*,* `fieldGoals`*,* `assists`*,* `freeThrows`*. Intuitively, this query obtains the skyline of good offensive players in the NBA.* ◇

4.2 Two-way SWJ queries over Data Streams

Through this scenario, we demonstrate how the LSJ operator handles skyline-window-join (SWJ) queries between two input data streams (Figure 2). The LSJ operator utilizes the *iteration-fabric* framework to run SWJ queries over the Intel Berkeley Research lab data streams. Attendees will have the opportunity to view the behaviour of the LSJ operator and observe the advantages that the *iteration-fabric* provides over other alternative solutions.

EXAMPLE 2 (LSJ OPERATION). *Consider a scenario in which a set of sensors produce readings only related to* `temperature` *and* `voltage`*, while another set of sensors give readings of* `humidity` *and* `light`*. This results in two input streams, namely* `stream-1 (moteid, temperature, voltage, epoch)` *and* `stream-2 (moteid, humidity, light, epoch)`*. Given these, a SWJ query over the set of sensors on the attributes* `temperature`*,* `voltage`*,* `humidity` *and* `light` *could be:*

```
Skyline = SWJ * FROM stream-1 S1, stream-2 S2,
                WHERE S1.moteid = S2.moteid,
                S1.epoch within last 24 hours,
                S2.epoch within last 24 hours,
                temperature MAX, voltage MAX,
                humidity MAX, light MAX.
```

This query returns a set of interesting *readings produced by the sensors over the past* 24 *hours.* ◇

4.3 Other Skyline-Join Operations

In this scenario, we demonstrate SkySuite's ability to process skyline-join queries over multiple data sources. Additionally, we also show how SkySuite handles a scenario in which one of the data sources is static, while the other is streaming. As show in Figure 2, SkySuite utilizes a hybrid form of the SSJ and LSJ operators to tackle skyline-join queries over hybrid input sources.

5. CONCLUSION

This demonstration introduces SkySuite: a framework of *skyline-join* operators that can be leveraged to efficiently process skyline-join queries over both static and stream environments. In particular, we demonstrate the Skyline-Sensitive Join (SSJ) and the Layered Skyline-window-Join (LSJ) operators. The SSJ operator overcomes the drawbacks of existing static skyline-join algorithms by pruning the join space in terms of blocks of data, as opposed to individual data points, thereby avoiding excessive point-to-point dominance checks. While, the LSJ operator provides an efficient technique for computing skyline-joins over pairs of streams. LSJ is first of its kind for answering *skyline-window-join* (SWJ) queries over data streams.

6. ACKNOWLEDGMENTS

We would like to thank Dr. Youngchoon Park of Johnson Controls, Inc. (JCI) for allowing us access to JCI data sets.

7. REFERENCES

[1] I. Bartolini, P. Ciaccia, and M. Patella. SaLSa: computing the skyline without scanning the whole sky. In *CIKM*, pages 405–414, 2006.
[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline operator. In *ICDE*, pages 421–430, 2001.
[3] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, pages 717–719, 2003.
[4] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski. Prefjoin: An efficient preference-aware join operator. In *ICDE*, pages 995–1006, 2011.
[5] H.-T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975.
[6] H.-G. Li, S. Chen, J. Tatemura, D. Agrawal, K. S. Candan, and W.-P. Hsiung. Safety guarantee of continuous join queries over punctuated data streams. In *VLDB*, pages 19–30, 2006.
[7] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE*, pages 502–513, 2005.
[8] M. Nagendra and K. S. Candan. Skyline-sensitive joins with LR-pruning. In *EDBT*, pages 252–263, 2012.
[9] M. Nagendra and K. S. Candan. Layered processing of skyline-window-join (SWJ) queries using iteration-fabric. In *ICDE*, 2013 (to be published).
[10] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.
[11] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application.* John Wiley, 1986.
[12] D. Sun, S. Wu, J. Li, and A. K. H. Tung. Skyline-join in distributed databases. In *ICDE Workshop*, pages 176–181, 2008.
[13] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, pages 301–310, 2001.
[14] Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. *TKDE*, 18(3):377–391, 2006.
[15] N. Tatbul and S. Zdonik. Window-aware load shedding for aggregation queries over data streams. In *VLDB*, pages 799–810, 2006.
[16] A. Vlachou, C. Doulkeridis, and N. Polyzotis. Skyline query processing over joins. In *SIGMOD*, pages 73–84, 2011.
[17] A. N. Wilschut and P. M. G. Apers. Dataflow query execution in a parallel main-memory environment. In *PDIS*, pages 68–77, 1991.