36th Int. Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM 2010)
(Invited Paper)

# Continuous Processing of Preference Queries in Data Streams

Maria Kontaki, Apostolos N. Papadopoulos, and Yannis Manolopoulos

Department of Informatics, Aristotle University
54124 Thessaloniki, Greece
{kontaki,papadopo,manolopo}@csd.auth.gr

**Abstract.** Preference queries have received considerable attention in the recent past, due to their use in selecting the most preferred objects, especially when the selection criteria are contradictory. Nowadays, a significant number of applications require the manipulation of time evolving data and therefore the study of continuous query processing has recently attracted the interest of the data management community. The goal of continuous query processing is to continuously evaluate long-running queries by using incremental algorithms and thus to avoid query evaluation from scratch, if possible. In this paper, we examine the characteristics of important preference queries, such as skyline, top-$k$ and top-$k$ dominating and we review algorithms proposed for the evaluation of continuous preference queries under the sliding window streaming model.

## 1 Introduction

Recently, preference queries have significantly attracted the research interest. Preference queries are frequently used in multicriteria decision making applications, where a number of (usually) contradictory criteria are involved to select the most convenient answers to the user.

Assume that a customer is interested in purchasing a PDA device. Assume further, that the customer focuses on two important characteristics of PDAs, namely, price and weight. Unfortunately, these two criteria are frequently contradictory and therefore, the number of candidates should be carefully selected. In this example, each PDA is represented as a tuple containing two attributes (price and weight) and the customer is interested in items that minimize these attributes. Depending on the semantics of each attribute, in other cases the customer may desire the maximization of the attributes in question, or any other combination. For the rest of the discussion, we consider that smaller values are preferable. However, most of the techniques mentioned in this paper are directly applicable to other cases as well.

Preference queries have received considerable attention in the past, due to their use in selecting the most preferred items, especially when the selection criteria are contradictory. Skyline [5, 8, 29, 44] and top-$k$ [3, 7, 10] queries have been thoroughly studied by the database community. Preferences have been used

previously in other disciplines such as Game Theory (e.g., Pareto optimality [34]) and Computational Geometry (e.g., maximal vectors [20]).

During the last years, we are witnessing a significant interest of the research community towards continuous query processing, due to the fact that many applications deal with data that change frequently with respect to time. In these types of applications, the goal is to continuously evaluate the query and report the result in real time contrary to ad-hoc query executions that are used in traditional applications [1, 12, 25]. Examples of such emerging applications are network monitoring, financial data analysis, sensor networks to name a few.

The most important property of data streams is that new tuples are continuously appended and, therefore, efficient storage and processing techniques are required to cope with high update rates. More specifically, a stream-oriented algorithm should satisfy the following requirements: (a) fast response time, (b) incremental evaluation, (c) limited number of data access, and (d) in memory storage to avoid expensive disk accesses. Therefore, algorithms proposed for traditional databases are not appropriate and new methods and techniques should be developed to fulfil the requirements posed by the data stream model.

Efficient stream processing algorithms are difficult to be designed, due to the unbounded nature of data streams. Several models have been proposed to reduce and bound the size of the streams. A class of algorithms focuses on the recent past of data streams by applying the *sliding window model* [2, 14]. This way, only the most recent tuples (*active* tuples) of the data stream are considered for query processing, whereas older tuples are not taken into account as they are considered obsolete.

There exist two basic forms of sliding windows. In a *count-based* sliding window, the number of active tuples remains constant (i.e., the sliding window contains the last $W$ tuples) and therefore for each new tuple that arrives, the oldest one expires. In a *time-based* sliding window, the number of active tuples may vary. The *expiration time* of a tuple does not depend on the arrival or expiration of other tuples. The set of active tuples is composed of all tuples arrived the last $T$ time instances. Figure 1 illustrates a count-based sliding window.

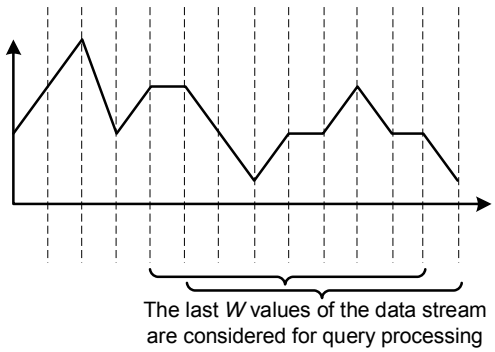



**Fig. 1.** Example of a count-based sliding window of length $W$=9.

In this paper, we discuss the state-of-the-art processing techniques of the most important preference queries in data streams under the sliding window model. More specifically, we study proposed methods for the continuous evaluation of skyline and top-$k$ queries. Moreover, we examine closely a new type of preference query, the top-$k$ dominating query, that recently has attracted significant researcher interest.

The rest of the paper is organized as follows. Sections 2 and 3 study different processing techniques of skyline and top-$k$ queries respectively and give detailed description for the most important ones. Section 4 presents the top-$k$ dominating query and discusses early related work on this topic. Finally, Section 5 concludes the work and discusses future research directions.

## 2 Skyline Queries

The skyline query is one of the most widely used preference queries. It is based on the *dominance relationship* between tuples. Assuming that smaller values are preferable in all dimensions, the dominance relationship and the skyline are defined as follows:

**Definition 1 (dominant tuple).** *A tuple $t_i$ dominates another tuple $t_j$ ($t_i \prec t_j$), if and only if $t_i$ is smaller than or equal to $t_j$ in all dimensions and it is strictly smaller than $t_j$ in at least one of them.*

**Definition 2 (skyline).** *The skyline consists of the tuples not dominated by any other tuple.*

The key advantage of the skyline query is that it does not require any user-defined information or parameter. Moreover, skyline queries are characterized by the scaling invariance property, which means that if scaling is applied to any dimension values, the result remains unchanged. On the other hand, as the dimensionality increases, the probability for a tuple to dominate another tuple is reduced significantly and, therefore, the number of skyline tuples increases substantially. Overall, the skyline query does not bound the size of the output and therefore in extreme cases it is possible that all the tuples be part of the skyline result.

In the past decade, skyline queries are thoroughly studied for several types of modern applications such as P2P networks [43], MANETs [15] and Web systems [4]. Moreover, methods for different types of data, such as spatial [39] and uncertain data [35], are developed to increase efficiency of skyline query processing. Additionally, several variations of the skyline have been proposed. In [44], the authors study the skyline computation in subspaces. Another variation has been proposed in [29] for selecting skyline tuples according to their domination capabilities. Dominance relationships between different data sets (e.g., products and customers) are examined in [27], where the authors proposed an organization scheme called DADA cube, to support a number of significant query types.

The first attempt to develop an algorithm more suitable for dynamic data was [41], where the authors presented a progressive algorithm for skyline computation. In the sequel, more sophisticated and efficient progressive methods have been developed [24, 37, 38]. These algorithms are not appropriate for continuous evaluation but, instead, they were developed to support online user preferences and data that are not updated frequently and only a small number of insertions/deletions can be handled efficiently. Table 1 contains a brief description of continuous skyline computation algorithms, which are studied in detail in the next subsections.

| **Method** | **Query Type** | **Window Type** | **Multiple Queries** |
|---|---|---|---|
| cnN | skyline | count-based | yes |
| LookOut | skyline | time-based | no |
| Lazy and Eager | skyline | both | no |
| Filter and Sampling | FSQW | time-based | no |
| CoSMuQ | k-dominant skyline | both | yes |

**Table 1.** Continuous skyline algorithms.

### 2.1 cnN

In [28], the authors proposed a novel technique for continuous skyline query processing. First, they provide a pruning technique to reduce the number of elements to be stored for processing. If the data distribution is independent and data values are always distinct, then the average number of elements that are stored is $O(\log^d N)$, where $N$ is the maximum size of the sliding windows and $d$ is the number of dimensions. The survived elements $R_N$ are organized in graphs based on the *critical dominance relationship* between them and these graphs form a forest. Assume two active elements $e_1$ and $e_2$. Element $e_1$ critical dominates $e_2$, if it is the youngest element that is older than $e_2$ and dominates it. Each critical dominance relationship is represented by an edge of the forest. The forest is encoded by a set of intervals of 1-dimensional space and then skyline query is evaluated by using stabbing queries.

In addition, a maintenance algorithm of $R_N$ and the encoded scheme of intervals is provided. Then, a trigger-based incremental algorithm *cnN* has been proposed to enable efficient continuous skyline processing. The proposed framework supports multiple $n$-of-$N$ skyline queries, i.e., skyline queries with different sliding window size $n$ ($n \leq N$). The continuous algorithm updates the result in $O(\delta)$ time per new data element and requires $O(\log s)$ time to update the trigger list per result change, where $\delta$ is the number of elements changed in the current result and $s$ is the number of skylines elements. The main disadvantage of the *cnN* algorithm is that it requires the maintenance of several structures, such as graphs, interval tree and $R^*$-tree used for the survived tuples.

### 2.2 LookOut

Algorithm *cnN* handles count-based sliding windows, whereas algorithm *LookOut* [33] has been proposed for time-based windows. *LookOut* uses a heap to store the skyline tuples and a $R^*$-tree to store the active tuples of the window. The proposed method utilizes some attractive properties of skyline queries, such as the *transitive* property. When a new tuple is inserted, we check if the new tuple is a skyline tuple or not. The algorithm uses a best-first search on the spatial index and discards nodes that their lower left corner does not dominate the new tuple. When a skyline tuple expires, we check the spatial index to find new skyline tuples. The key observation is that new skyline tuples should be part of the skyline of the space dominated by the expired tuple. These tuples are detected by a best-first search and then are checked further to discard tuples that are dominated by other skyline tuples. Notice that the proposed algorithm does not prune tuples (all the active tuples are stored in a $R^*$-tree), although this is possible as we will see in the next subsection.

### 2.3 Lazy and Eager

In [42], two algorithms have been presented, called *Lazy* and *Eager*. We begin our description with the first one. When a new tuple $t$ arrives, *Lazy* searches for tuples in the *dominance region* (DR) and the *antidominance region* of $t$ to update the skyline tuples. Region $t.DR$ is the data space that is dominated by $t$, whereas $t.ADR$ is the area where a tuple dominating $t$ could fall. When a skyline tuple expires, the algorithm deletes all the expired tuples (notice that *Lazy* keep stored tuples even after their expiration) and then tries to find possible skyline tuples in the area dominated *exclusively* by the expired skyline tuple. *Lazy* and *LookOut* methods are similar, since they have same operations.

To improve performance, two attractive properties of "stream skylines" have been presented in [42]: (a) a tuple can be safely discarded if it is dominated by an incoming tuple, and (b) a tuple can be part of skyline for at most a single continuous time interval, which is called *skyline influence time* and denotes the minimum possible time at which the tuple can be part of skyline. Algorithm *Eager* utilizes these properties. Therefore, when a new tuple $t$ is inserted, *Eager* during the skyline update also discards the active tuples belonging to $t.DR$. Then, it computes the skyline influence time of $t$ and forms an event that will be processed in this time. *Eager* compared to *Lazy* reduces the memory consumption by keeping those tuples that may be inserted in the skyline, but requires additional overhead to process the events.

### 2.4 Filter and Sampling

Continuous skyline queries report the skyline of the active tuples in each update. These types of skylines are called *snapshot skylines*. In [47], the authors state that snapshot skylines are not very meaningful, since in a streaming environment the skyline tuples may change too fast, and thus, it will be more interesting to

identify tuples that are consistently part of skyline. Therefore, they introduce a variation of snapshot skylines called *frequent skyline query over a sliding window* (FSQW), which returns the tuples appearing in the skylines of at least $\mu$ of the $n$ most recent timestamps. It is proved that an exact algorithm for FSQW requires the exact snapshot skyline computation.

The server-client architecture is considered, where the server continuously maintains the result. Moreover, changes to existing tuples are considered instead of new tuples, i.e., each client transmit a specific tuple and changes of it. Three algorithms have been proposed aiming at minimizing the communication overhead instead of processing cost [47]. The first algorithm, called $Filter$, is an exact algorithm and therefore can be also used for the evaluation of snapshot skylines. $Filter$ avoids the transmission of updates to the server if they cannot influence the skyline. Specifically, the server computes a filter for each tuple and an update is transmitted from the client to the server only if (a) the tuple violates its filter, or (b) the server specifically asks the tuple. There are many possible filters for each tuple. The authors propose a model that tries to balance the transmissions due to filter violations and server requests.

Although $Filter$ reduces significantly the number of updates transmitted to the server, its performance may degrade for large and frequently updated data sets. Algorithm $Sampling$ has been proposed to reduce the communication overhead by computing approximate FSQW outputs. According to $Sampling$, all clients report their current status with some global probability $R$, which depends on the trade-off between user-defined accuracy and overhead. Finally, an algorithm, called $Hybrid$, has been presented that combines $Filter$ and $Sampling$. $Hybrid$ exploits the advantages of the two algorithms and avoids their disadvantages by allowing records to switch among different modes.

### 2.5 CoSMuQ

The number of skyline tuples depends heavily on the dimensionality of the data set and the data distribution. The number of tuples in the skyline increases substantially with increasing dimensionality, leading to difficulties in selecting the best object that satisfy user's preferences. Towards eliminating the huge number of skyline tuples, a novel method has been proposed in [11], which relaxes the dominance definition to increase the probability that a tuple will dominate another. Evidently, by increasing this probability, the number of skyline tuples is reduced. Instead of searching for ordinary skyline tuples in all dimensions, $k$-dominant skylines are being used. Assuming that the smaller values are preferable, the following definitions explain:

**Definition 3 ($k$-dominant tuple).** *A tuple $t_i$ $k$-dominates another tuple $t_j$, if and only if $t_i$ is smaller than or equal to $t_j$ in at least $k$ dimensions and it is strictly smaller than $t_j$ in at least one of them.*

**Definition 4 ($k$-dominant skyline).** *The $k$-dominant skyline consists of the tuples not $k$-dominated by any other tuple.*

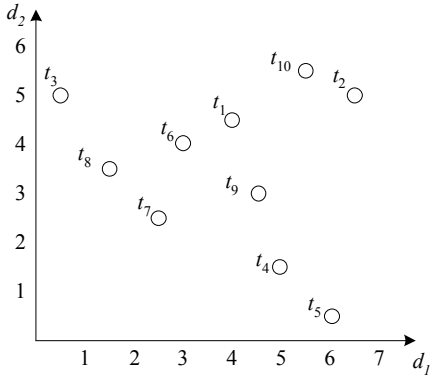
The processing techniques reported in [11] are executed in static data sets. In [21] the continuous $k$-dominant skyline evaluation over sliding windows is studied. The proposed algorithm $CoSMuQ$ handles multiple continuous queries. Each query may be defined in a subset of the available dimensions, since different users are usually interested in different attributes. Moreover, the parameter $k$ set by each query, may be different. The proposed method divides the space in pairs of dimensions. For each pair, a grid is used to compute skyline tuples for these dimensions. Then, the method exploits the discovered skyline tuples to eliminate candidate $k$-dominant skyline tuples and it combines the partial results to compute the final result. The proposed scheme uses only simple domination checks, which are faster than $k$-dominant checks. However, in high dimensional spaces, $CoSMuQ$ considers a large number of grids.

## 3 Top-$k$ Queries

A top-$k$ query uses a user-defined *preference function* to assign scores to tuples and rank them. Assuming that smaller values of the preference function are preferable, the top-$k$ query is defined as follows:

**Definition 5 (top-$k$).** *Given a data set and a preference function $F$, a top-$k$ query returns the $k$ tuples in the data set with the smallest scores according to $F$.*

Figure 2 depicts a time instance of the sliding window ($W$=10) of two-dimensional tuples. Assume that the preference function $F$ is the sum of the values in two dimensions, whereas smaller values of $F$ are preferable. A top-3 query ($k$=3) retrieves tuples $t_7$, $t_8$ and $t_3$, since they have the three smallest score 5, 5 and 5.5 respectively. In this example, the skyline consists of tuples $t_3$, $t_4$, $t_5$, $t_7$ and $t_8$, since these tuples are not dominated by any other active tuple.




(a) 2-dimensional representation

| *id* | $d_1$ | $d_2$ | $F=d_1+d_2$ |
|---|---|---|---|
| $t_1$ | 4 | 4.5 | 8.5 |
| $t_2$ | 6.5 | 5 | 7 |
| $t_3$ | 0.5 | 5 | 5.5 |
| $t_4$ | 5 | 1.5 | 6.5 |
| $t_5$ | 6 | 0.5 | 6.5 |
| $t_6$ | 3 | 4 | 7 |
| $t_7$ | 2.5 | 2.5 | 5 |
| $t_8$ | 1.5 | 3.5 | 5 |
| $t_9$ | 4.5 | 3 | 7.5 |
| $t_{10}$ | 5.5 | 5.5 | 11 |

(b) tuples and scores

**Fig. 2.** Example of preference queries.

In contrast to skyline queries, top-$k$ queries bound the output size. If two or more tuples have the same score, then we can either: (a) report all these tuples but we may expect more than $k$ tuples in the result, or (b) use a tie-breaking criterion, e.g., the value of a specific dimension. The major disadvantage of the top-$k$ query is that it requires a user-defined preference function. This means that different preference functions can lead to different score assignments and therefore in different results. Thus, the analysis of the results is not straightforward. Moreover, it is not always easy for a user to specify the appropriate preference function, especially with growing number of dimensions.

The literature is rich in methods proposed for the efficient evaluation of top-$k$ queries [3, 7, 10, 16]. Top-$k$ queries have been studied in the context of several database types, such as relational [3], multimedia [9] and web databases [31]. Algorithms *Onion* [7] and $Prefer$ [16] have been proposed for top-$k$ queries in traditional databases. *Onion* uses the convex hulls of the database, whereas $Prefer$ uses sorted lists. In [10], the authors presented algorithm $MPro$ for the optimization of expensive predicates processing. Views are used in [13] to answer top-$k$ queries. All the aforementioned methods are appropriate for conventional databases. There are many other methods in the bibliography but their description is beyond the scope of this paper. An excellent survey on top-$k$ query processing in relational databases can be found in [18]. We continue with the detailed description of methods proposed for continuous top-$k$ evaluation. A summary of the studied algorithms is given in Table 2.

| **Method** | **Query Type** | **Window Type** | **Multiple Queries** |
|---|---|---|---|
| TMA and SMA | top-$k$ | both | yes |
| Distributed top-$k$ | distributed top-k | time-based | no |
| Compact Set based | top-$k$ on uncertain data | time-based | no |
| Det and Sam | top-$k$ on uncertain data | time-based | no |

**Table 2.** Continuous top-$k$ algorithms.

### 3.1 TMA and SMA

Mouratidis et al. [32] proposed two algorithms for continuous top-$k$ query execution in sliding windows. The first one, called $TMA$, is based on simple observations regarding the continuous processing of top-$k$ queries. Each top-$k$ query has an *influence region*. The influence region of a query determines the data space in which a tuple should belong so that it may change the query result. When a new tuple is inserted, $TMA$ checks if the new tuple belongs to the influence region of any query and updates its top-$k$ respectively. When a top-$k$ tuple expires, a from scratch computation is performed if the new tuple does not have better score than the expired tuple.

To overcome this disadvantage, the authors proposed the algorithm $SMA$, which is based on the observation that the records appearing in some result of top-$k$ are the ones that belong to the $k$-skyband [38] in the score-time space. Thus, the proposed algorithm transforms the problem of continuous top-$k$ into $k$-skyband maintenance. $SMA$ restricts the skyband maintenance for a query to tuples falling inside its influence region. It uses a balanced tree to store the arrival time of the $k$-skyband tuples sorted in descending order. Moreover, it keeps the *dominance counter* for each tuple $t$ stored in balanced tree, which is the number of tuples with better score that arrive after $t$. However, there may be a scenario where the skyband contains fewer than $k$ tuples. In such a case, the algorithm computes the skyband tuples from scratch.

### 3.2 Distributed Top-$k$

Babcock and Olston [6] proposed a distributed algorithm for top-$k$ query processing. The problem considered in [6], assumes a central node and a number of monitor nodes, each one monitoring a data object. The changes of a data object form a data stream and, therefore, multiple data streams should be processed by the central node. The transmission of the entire data stream from a monitor node to the central node is unnecessary. Thus, the authors proposed a scheme in which arithmetic constraints are maintained at monitor nodes and distributed communication happens when constraints are violated. Then, the central node updates the top-$k$ result and assigns new constraint to the monitor node.

The proposed algorithm is exact, i.e., the central node has the correct top-$k$ output in every time instance. Additionally, approximate answers have been studied. Moreover, the algorithm aims at minimizing the network overhead. However, the proposed scheme is not fully distributed, since it uses some sort of "base station" or "coordinator".

### 3.3 Compact Set-based Algorithms

In [19], the processing of continuous top-$k$ queries in a sliding window over uncertain data streams is examined. The challenge of processing queries on uncertain data streams lies on high update rates and the exponential growth in the number of possible worlds induced by the uncertain data model. The paper adopts a simple uncertain data model, where each tuple appears with a certain probability independent of other tuples. Also, it defines the $Pk-topk$ query that returns the $k$ most probable tuples of being the top-$k$ among all. Moreover, the paper introduces the concept of *compact set* on which the proposed synopses are based on.

The authors extend the problem of uncertain top-$k$ queries on static data sets to the case of data streams over sliding windows. First, a synopsis is presented that can handle only insertions (i.e., landmark windows). However, handling deletions is much more difficult and requires carefully designed synopses. The paper proposed several space and time efficient synopses with provable bounds

to enable continuous top-$k$ evaluation over sliding windows. Overall, the authors proposed and evaluated five algorithms based on the compact set concept.

### 3.4 Det and Sam

Hua and Pay [17] also examined the continuous evaluation of top-$k$ queries over uncertain data streams. They proposed a novel uncertain data stream model and introduced the continuous probabilistic threshold top-$k$ queries. More specifically, given a probabilistic threshold top-$k$ query, a set of uncertain data streams and a sliding window length, the continuous probabilistic threshold top-$k$ query reports the set of uncertain data streams whose top-$k$ probabilities in the sliding window are at least $p$, for each time instant $t$.

The authors proposed four algorithms. The exact algorithm, called *Det*, computes the exact answer of a continuous probabilistic threshold top-$k$ query. Moreover, they proposed a sampling algorithm, called *Sam*, which estimates the probability that an uncertain object being ranked top-$k$ via sampling. Probabilistic guarantees are also provided. Then, *Sam* computes an approximation answer based on the estimated probabilities. Additionally, quantile summary techniques are applied to develop the space efficient versions of both algorithms.

## 4 Top-$k$ Dominating Queries

Recently, an interesting alternative has been proposed [38], which combines the dominance concept with the notion of scoring functions. This new query is called top-$k$ dominating query. The following definitions explain:

**Definition 6 (domination power).** *The domination power of a tuple is the number of tuples it dominates.*

**Definition 7 (top-$k$ dominating query).** *A top-k dominating query retrieves the k tuples in the data set with the highest domination power.*

To clarify the definitions above, assume the example of Figure 2. A top-3 ($k$=3) dominating query retrieves tuples $t_7$, $t_8$ and $t_6$ with domination power 5, 4 and 3 respectively. All the other tuples have smaller domination power than 3. Although preference queries as skyline and top-$k$, have been studied in a data stream perspective, top-$k$ dominating queries have not received adequate attention for this scenario. However, the top-$k$ dominating query is an important decision support tool. In a sense, it combines skyline and top-$k$ queries, resulting in a more complex one. Practically, it preserves their advantages without sharing their limitations. Top-$k$ dominating queries use the dominant relationship rather than a user defined score function. The determination of the appropriate score function is not obvious, especially when the number of attributes increases. On the other hand, top-$k$ dominating queries use an intuitive score to rank the tuples that can be interpreted easily by a non-expert. Moreover, top-$k$ dominating queries bound the size of the resulting set of tuples, in contrast to skyline

queries, where the size of the result is unbounded and increases significantly as the number of attributes grows. Additionally, they preserve the scaling invariance property.

In skylines and top-$k$ queries, we can use the expiration time to prune tuples resulting in more efficient algorithms with respect to the memory requirements and the response time. More specifically, in a skyline query, if a tuple $t_i$ is dominated by another tuple $t_j$ and expires before $t_j$, then it is safe to prune tuple $t_i$. Assume the example of Figure 2. Assume further that the pointer of a tuple denotes its arrival time, i.e., $t_1.arr$=1. Tuple $t_1$ is dominated by $t_8$ end expires before $t_8$, thus $t_1$ can be safely discarded. In a top-$k$ query, if $k$ tuples exist with better scores than the score of a tuple $t_i$ and $t_i$ expires before them, then it is safe to discard tuple $t_i$. Returning to the example of Figure 2, the score of $t_1$ is 8.5 and there are more than 3 tuples with score better than 8.5 and expire after $t_1$ (e.g., $t_3, t_4$ and $t_5$), therefore $t_1$ can be pruned. On the other hand, top-$k$ dominating queries are more complicated. It is not possible to discard tuples, even if we know that it is not possible to be in the result during their lifespan. This is because the existence of a tuple affects the domination power of the other tuples.

Top-$k$ dominating queries have been addressed in [45, 46, 26]. In [45, 46] the authors proposed efficient algorithms to determine the top-$k$ dominating tuples by using an aggregate R-tree index. In [26], the authors studied efficient algorithms for top-$k$ dominating query processing in uncertain databases. A pruning approach has been proposed to reduce the space of a probabilistic top-$k$ dominating query and in addition, approximate queries are examined. The domination power is also used in [36] to rank multidimensional tuples. The concept of dominance score has been used by [40] towards ranking web services.

The literature is limited regarding continuous variants of top-$k$ dominating query processing techniques over data streams. In [23], the authors use a simple grid-based indexing scheme to facilitate efficient search and update operations avoiding expensive reorganization costs of dynamic hierarchical access methods. Three algorithms are proposed. $BFA$ is a naive approach computing all the domination checks in each update and it is simply proposed for comparison reasons. The algorithms $EVA$ and $ADA$ use an event-based approach to reduce both the number of domination checks during an update and the number of exact score computations as well. $ADA$ use two optimizations regarding the event computations achieving the decrease of the number of events processed and, therefore, enhancing the efficiency of $EVA$. Subspace top-$k$ dominating queries are examined in [22]. A new grid-based indexing scheme is proposed, called adaptive grid, to efficiently process subspace top-$k$ dominating queries. Moreover, several optimizations are proposed to enhance the query processing mechanism. These methods are the first attempt for continuous top-$k$ dominating queries evaluation.

## 5 Conclusions and Future Work

In this paper, we review various preference queries and we discuss their differences. Moreover, we thoroughly examine the related work on continuous processing of preference queries over sliding windows and we discuss the advantages and the disadvantages of the proposed methods.

Skyline queries have been examined more than other preference queries in the context of data streams. Various methods have been proposed to handle both count-based and time-based sliding windows. However, there are many open issues to be addressed. Subspace skyline queries have not been studied as well as many widely used variations of skylines such as constrained skylines. Recently, a theoretical study of skyline cardinality estimation over sliding windows is presented in [30]. The authors estimate skyline cardinality over uniformly and arbitrary distributed data. The results of this study can be used to optimize the query methodology to improve memory consumption and processing cost or the network overhead.

Although the study of continuous top-$k$ queries has moved onto uncertain data streams, there are many research directions that should be examined, to develop more sophisticated and efficient methods for continuous top-$k$ queries in certain data. Methods, more efficient than the existing ones, are required to handle the expiration of tuples. Distributed top-$k$ evaluation can be enhanced further by developing methods that have as uniformly as possible energy consumption. Methods can exploit various characteristics of data types such as spatial data to improve efficiency in specific modern applications.

Recently, a new alternative preference query, the top-$k$ dominating query, has attracted the research interest. This query is an important tool for decision support since it provides data analysts an intuitive way for finding significant objects. However, the related work is very limited and therefore there are many issues on this topic to be addressed. For example, approximate evaluation is a very promising research direction, since in many cases we are willing to trade accuracy for speed of computation. More complicated scenarios can be also examined, such as distributed environments or multiple queries existence.

## References


1. C.C. Aggarwal: "Data streams: Models and algorithms", *Springer*, 2007.
2. B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom: "Models and Issues in Data Stream Systems", *In Proc. of PODS*, pp.1-16, 2002.
3. N. Bruno, S. Chaudhuri and L. Gravano: "Top-$k$ selection queries over relational databases: Mapping strategies and performance evaluation", *ACM TODS*, Vol.27, No.2, pp.153-187, 2002.
4. W.-T. Balke, U. Guntzer and J.X. Zheng: "Efficient distributed skylining for web information systems", *In Proc. of EDBT*, pp.256-273, 2004.
5. S. Borzsonyi, D. Kossmann and K. Stocker: "The skyline operator", *In Proc. of ICDE*, pp.421-430, 2001.
6. B. Babcock and C. Olston: "Distributed top-$k$ monitoring", *In Proc. of SIGMOD*, pp.28-39, 2003.

7. Y.-C. Chang, L.D. Bergman, V. Castelli, C.-S. Li, M.-L. Lo and J.R. Smith: "The Onion technique: Indexing for linear optimization queries", *In Proc. of SIGMOD*, pp.391-402, 2000.
8. J. Chomicki, P. Godfrey, J. Gryz and D. Liang: "Skyline with presorting", *In Proc. of ICDE*, pp.717-719, 2003.
9. S. Chaudhuri, L. Gravano and A. Marian: "Optimizing top-$k$ selection queries over multimedia repositories", *IEEE TKDE*, Vol.16, No.8, pp.992-1009, 2004.
10. K.C.-C. Chang and S. Won Hwang: "Minimal probing: Supporting expensive predicates for top-$k$ queries", *In Proc. of SIGMOD*, pp.346-357, 2002.
11. C.Y. Chan, H.V. Jagadish, K.-L. Tan, A.K.H. Tung and Z. Zhang: "Finding $k$-dominant skylines in high dimensional space", *In Proc. of SIGMOD*, pp.503-514, 2006.
12. N. Chaudhry, K. Shaw and M. Abdelguerfi: "Stream data management", *Springer*, 2006.
13. G. Das, D. Gunopulos, N. Koudas and D. Tsirogiannis: "Answering top-$k$ queries using views", *In Proc. of VLDB*, pp.451-462, 2006.
14. J. Gehrke, F. Korn and D. Srivastava: "On computing correlated aggregates over continual data stream", *ACM SIGMOD Record*, Vol.30, No.2, pp.13-24, 2001.
15. Z. Huang, C.S. Jensen, H. Lu and B.C. Ooi: "Skyline queries against mobile lightweight devices in MANETs", *In Proc. of ICDE*, pp.66, 2006.
16. V. Hristidis and Y. Papakonstantinou: "Algorithms and applications for answering ranked queries using ranked views", *VLDB Journal*, Vol.13, No.1, pp.49-70, 2004.
17. M. Hua and J. Pei: "Continuously monitoring top-$k$ uncertain data streams: a probabilistic threshold method", *Distrib. Parallel Databases*, Vol.26, No., pp.2965, 2009.
18. I.F. Ilyas, G. Beskales and M.A. Soliman: "A survey of top-$k$ query processing techniques in relational database systems", *ACM Comput. Surv.*, Vol.40, No.4, 2008.
19. C. Jin, K. Yi, L. Chen, J.X. Yu and X. Lin: "Sliding window top-$k$ queries on uncertain streams", *In Proc. of PVLDB*, pp.301-312, 2008.
20. H.T. Kung: "On finding the maxima of a set of vectors", *Journal of the ACM*, Vol.22, No.4, pp. 469-476, 1975.
21. M. Kontaki, A.N. Papadopoulos and Y. Manolopoulos: "Continuous $k$-Dominant Skyline Computation on Multidimensional Data Streams", *In Proc. of SAC*, pp.16-20, 2008.
22. M. Kontaki, A.N. Papadopoulos, and Y. Manolopoulos: "Continuous Top-k Dominating Queries in Subspaces", *In Proc. of PCI*, 2008.
23. M. Kontaki, A.N. Papadopoulos and Y. Manolopoulos: "Continuous top-$k$ dominating queries", Technical Report, Aristotle University of Thessaloniki, 2009.
24. D. Kossmann, F. Ramsak and S. Rost: "Shooting stars in the sky: an online algorithm for skyline queries", *In Proc. of VLDB*, pp.275-286, 2002.
25. N. Koudas and D. Srivastava: "Data stream query processing: a tutorial", *In Proc. of VLDB*, pp.1149, 2003.
26. X. Lian and L. Chen: "Top-$k$ dominating queries in uncertain databases", *In Proc. of EDBT*, pp.660-671, 2009.
27. C. Li, B.C. Ooi, A.K.H. Tung and S. Wang: "DADA: a data cube for dominant relationship analysis", *In Proc. of SIGMOD*, pp.659-670, 2006.
28. X. Lin, Y. Yuan, W. Wang and H. Lu: "Stabbing the sky: Efficient skyline computation over sliding windows", *In Proc. of ICDE*, pp.502-513, 2005.
29. X. Lin, Y. Yuan, Q. Zhang and Y. Zhang: "Selecting stars: the $k$ most representative skyline operator", *In Proc. of ICDE*, pp.86-95, 2007.

30. Y. Lu, J. Zhao, L. Chen, B. Cui and D. Yang: "Effective skyline cardinality estimation on data streams", *In Proc. of DEXA*, pp.241-254, 2008.
31. A. Marian, N. Bruno and L. Gravano: "Evaluating top-$k$ queries over web-accessible databases", *ACM TODS*, Vol.29, No.2, pp.319-362, 2004.
32. K. Mouratidis, S. Bakiras and D. Papadias: "Continuous monitoring of top-$k$ queries over sliding windows", *In Proc. of SIGMOD*, pp.635-646, 2006.
33. M.D. Morse, J.M. Patel and W.I. Grosky: "Efficient continuous skyline computation", *In Proc. of ICDE*, pp.108, 2006.
34. M.J. Osborne and A. Rubenstein: "A course in game theory", *MIT Press*, 1994.
35. J. Pei, B. Jiang, X. Lin and Y. Yuan: "Probabilistic skylines on uncertain data", *In Proc. of VLDB*, pp.15-26, 2007.
36. A.N. Papadopoulos, A. Lyritsis, A. Nanopoulos and Y. Manolopoulos: "Domination Mining and Querying", *In Proc. of DaWaK*, pp.145-156, 2007.
37. D. Papadias, Y. Tao, G. Fu and B. Seeger: "An optimal and progressive algorithm for skyline queries", *In Proc. of SIGMOD*, pp.467-478, 2003.
38. D. Papadias, Y. Tao, G. Fu and B. Seeger: "Progressive skyline computation in database systems", *ACM TODS*, Vol.30, No.1, pp.41-82, 2005.
39. M. Sharifzadeh and C. Shahabi: "The spatial skyline queries", *In Proc. of VLDB*, pp.751-762, 2006.
40. D. Skoutas, D. Sacharidis, A. Simitsis, V. Kantere and T. Sellis: "Top-$k$ Dominant Web Services Under Multi-Criteria Matching", *In Proc. of EDBT*, pp.898-909, 2009.
41. K.L. Tan, P.K. Eng and B.C. Ooi: "Efficient progressive skyline computation", *In Proc. of VLDB*, pp.301-310, 2001.
42. Y. Tao and D. Papadias: "Maintaining sliding window skylines on data streams", *IEEE TKDE*, Vol.18, No.3, pp.377-391, 2006.
43. S. Wang, B.C. Ooi, A.K.H. Tung and L. Xu: "Efficient skyline query processing on peer-to-peer networks", *In Proc. of ICDE*, pp.1126-1135, 2007.
44. T. Xia and D. Zhang: "Refreshing the sky: The compressed skycube with efficient support for frequent updates", In Proc. of SIGMOD, pp.491-502, 2006.
45. M.L. Yiu and N. Mamoulis: "Efficient processing of top-$k$ dominating queries on multi-dimensional data", *In Proc. of VLDB*, pp.483-494, 2007.
46. M.L. Yiu and N. Mamoulis: "Multidimensional top-$k$ dominating queries", *VLDB Journal*, Vol.18, No.3, pp.695-718, 2009.
47. Z. Zhang, R. Cheng, D. Papadias and A.K.H. Tung: "Minimizing the communication cost for continuous skyline maintenance", *In Proc. of SIGMOD*, pp.495-508, 2009.