Continuous Top-k Dominating Queries

Maria Kontaki, Apostolos N. Papadopoulos, and Yannis Manolopoulos

Abstract—Top-k dominating queries use an intuitive scoring function which ranks multidimensional points with respect to their dominance power, i.e., the number of points that a point dominates. The k points with the best (e.g., highest) scores are returned to the user. Both top-k and skyline queries have been studied in a streaming environment, where changes to the data set are very frequent. In such an environment, continuous query processing techniques are required toward efficient monitoring of query results, since periodic query re-execution is computationally intensive, and therefore, prohibitive. This work contains the first study of continuous top-k dominating queries over data streams. In comparison to continuous top-k and skyline queries, continuous top-k dominating queries pose additional challenges. Three exact algorithms (BFA, EVA, ADA) are studied, and among them ADA, which is enhanced with additional optimization techniques, shows the best overall performance. In some cases, we are willing to trade accuracy for speed. Toward this direction, two approximate algorithms are proposed (AHBA and AMSA). AHBA offers probabilistic guarantees regarding the accuracy of the result based on the Hoeffding bound, whereas AMSA performs a more aggressive computation resulting in more efficient processing. Evaluation results, based on real-life and synthetic data sets, show the efficiency and scalability of our techniques.

Index Terms—Top-k dominating queries, data streams, continuous queries, algorithms, analysis, approximation.

1 Introduction

During the last decade, we are witnessing a significant interest of the data management community in *preference-based query processing* [12], [14] where in addition to hard constraints (e.g., price $< 100€$) the results must satisfy some additional specific properties (known as preferences) related to the attribute values associated with each tuple. Tuples are typically represented as points in $\mathbb{R}^d$ space, where d is the number of attributes. For the rest of the discussion, we use the term *point* to denote a tuple and the term *dimension* to denote an attribute.

Two of the most widely used preference-based queries are: 1) the *top-k query* and 2) the *skyline query*. In a top-k query, a ranking function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is required, which assigns a value to each point p. The result of a top-k query comprises the k points with the highest values with respect to $f()$. A nice feature of top-k queries is that the number of answers is controlled by the parameter k, although in some cases the cardinality of the result set may exceed k due to ties (i.e., two or more points may have the same value.) In such a case, either all ties will be part of the answer or a tie-breaking criterion is applied to select exactly k answers. The most important limitation of top-k queries is that a ranking function is required. This function is usually user defined, whereas different functions generally result in different results. Moreover, in several cases the selection of an appropriate ranking function is not intuitive. For example, in an e-commerce application, there is no straightforward way to combine the attributes *CPU speed* and *battery autonomy* to select the most interesting laptop computers.

- *The authors are with the Department of Informatics, Aristotle University, Thessaloniki 54124, Greece.*
E-mail: {kontaki, papadopo, manolopo}@csd.auth.gr.

Manuscript received 11 Dec. 2009; revised 15 Oct. 2010; accepted 29 Nov. 2010; published online 4 Feb. 2011.
Recommended for acceptance by B.C. Ooi.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2009-12-0838.
Digital Object Identifier no. 10.1109/TKDE.2011.43.

Skyline queries, on the other hand, do not require a ranking function and they have the scaling invariance property, meaning that if scaling is applied to dimension values the result remains unchanged. The result of a skyline query is composed of the points that are not *dominated* by any other point. The dominance relationship depends on the semantics of each attribute; in some cases, small values are preferable (e.g., price) whereas in other cases large values are desirable (e.g., quality). Without loss of generality, we focus on minimizing dimension values (the smaller the better). Therefore, a point p dominates another point q ($p \prec q$), if and only if p is no worse than q in all dimensions and strictly better than q in at least one dimension.

Recently, an interesting alternative has been proposed which combines the concept of dominance with the notion of ranking functions. This new query is termed *top-k dominating query* [25], [36], [37] and in a sense it is a combination of top-k and skyline queries: it uses a ranking function to rank points (as in top-k queries) and it uses the dominance relationship (as in skyline queries). The score associated with a point p_i, denoted as $score(p_i)$, equals the number of points that p_i dominates. The motivation behind this idea is to define a preference query that maintains the advantages and eliminates the limitations of both top-k and skyline queries. Therefore, top-k dominating queries have the following desirable properties:

1. the number of results is controllable,
2. the result is scaling invariant,
3. no user-defined ranking function is required, and
4. each point is assigned an intuitive score which determines its rank.

To illustrate the differences among the aforementioned query types, we give a simple example. Let $\mathcal{P}$ be a set of 2-dimensional points corresponding to records with two attributes as shown in Fig. 1. The coordinates of each point are given in parentheses. According to the previous discussion, the skyline set comprises all points that are

p	$f(p)$	$score(p)$
p_1	10	2
p_2	12	0
p_3	7	1
p_4	7	3
p_5	7	2
p_6	8	3
p_7	5	6
p_8	5	5
p_9	7	3
p_{10}	12	0

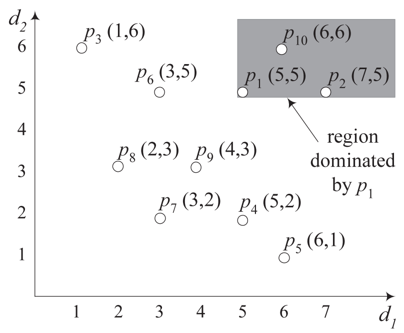


Fig. 1. Points and scores.

not dominated. These points are p_3, p_8, p_7, and p_5. Let $f(p) = p.x + p.y$ be a monotonically increasing ranking function. An example top-k query could ask for the $k = 2$ points with the best (lowest) rank. In this case, the preferred points are p_7 and p_8. Finally, a top-3 dominating query would return p_7, p_8, and p_6,[1] since these points provide the highest domination score (the number of points they dominate is the largest).

Assume that the network operation center of a University campus is interested in continuously detecting computers that may establish possible attacks. Unfortunately, there are a lot of criteria that should be balanced before a wise choice is made. Assume further that three characteristics of each computer are used: 1) the consumed bandwidth, 2) the number of connections, and 3) the number of destinations of its connections. A skyline query will return computers that tend to maximize one of the above attributes. Additionally, the size of the result is not bounded. Moreover, the design of the appropriate ranking function for a top-k query can be turned out very difficult, even for a network analyst. On the other hand, a continuous top-k dominating query on the 3-dimensional space is a more intuitive way to detect computers with suspicious behavior.

Efficient algorithms have been proposed to answer top-k queries, skyline queries, and, more recently, top-k dominating queries. The fundamental property of these algorithms is that they operate in an ad hoc fashion, meaning that they initiate a query processing task only if a query is issued. This is sufficient for applications operating on static or almost static data sets, where updates are rare. However, many modern applications adopt the streaming model of computation, and therefore, *continuous* query processing algorithms are required to refresh the query result. Examples of such emerging applications are computer network monitoring, scientific data analysis, data management in sensor networks, document filtering in information retrieval, web-based alerts, publish/subscribe services, just to name a few. The common property of these applications is that updates are very frequent, rendering query reexecution a nonviable solution.

This paper contains the first study of continuous top-k dominating queries over multidimensional data streams. We adopt the *sliding window* approach [1], where only the n most recent points, called *active points*, are taken into account. We associate to each point p_i two time instances: $p_i.arr$ is the arrival time of p_i, whereas $p_i.exp$ is the corresponding expiration time. When a point expires, it is removed from the set of active points.

Continuous query evaluation poses significant challenges in comparison to ad hoc processing. In continuous skyline queries, if p_i is dominated by p_j and p_j expires later than p_i (i.e., $p_j.exp > p_i.exp$), then it is safe to prune p_i since it will never be part of the skyline because of p_j. Moreover, in skyline queries, the transitivity property holds, i.e., if p_j dominates p_i and p_i dominates p_h then p_j dominates p_h. Thus, if p_i is discarded, one can still discard p_h due to the existence of p_j, as long as p_j expires later than both p_i and p_h. In a continuous top-k query, if there are k points with a better rank than p_i, and p_i expires earlier, it is safe to discard p_i. Moreover, the rank of a point does not vary as time progresses and it is not affected by other points, since the function $f()$ is based on the attribute values of each point. Therefore, one can discard points if the above condition holds without affecting the query result.

On the other hand, continuous top-k dominating queries are much more complicated regarding pruning and processing. First, it is not possible to eliminate a point, even if it will never be part of the top-k result. This is because the existence of a point affects the domination score of others and thus, the arrival/expiration of a point may change the score of a significant number of other points. Second, the computation of the domination score of a point is a costly operation and it must be avoided whenever possible. However, incoming points must be assigned a score in order to decide if they are part of the answer. Existing algorithms are not equipped with the necessary tools to handle these challenges. On the contrary, our algorithms are able to continuously monitor the result of top-k dominating queries efficiently. The key contributions of this work have as follows:

- We study three exact algorithms for continuous top-k dominating query processing, one of which is the clear winner due to its excellent performance. The proposed algorithm is based on a carefully designed event scheduling technique, toward avoiding costly computations.
- A performance analysis is offered studying the expected number of the most expensive computations incurred.
- Two approximate algorithms are proposed. These algorithms introduce a very useful efficiency/accuracy tradeoff, and on average they achieve more than 95 percent accuracy, being between one and three orders of magnitude faster than the best exact algorithm.
- A thorough experimental evaluation is carried out based on real-life and synthetic data sets, providing evidence regarding the applicability, scalability, and efficiency of the proposed algorithms.
- A discussion is carried out regarding the support of alternative sliding window models.

The rest of the paper is organized as follows. Section 2 discusses related work in the area. Section 3 offers background information and defines the problem. Section 4

1. Note that p_4 and p_9 have the same score as p_6. Optionally, the system could return these points also. If we allow this, then we should expect that the number of answers may be larger than k. Alternatively, a tie-breaking criterion may be used to discard points.

studies the properties of continuous top-k dominating queries and proposes a sliding window algorithm to solve the problem. Section 5 introduces some optimizations for performance improvement. Section 6 performs a theoretical analysis of the proposed algorithm. Approximate algorithms are studied in Section 7, offering faster execution times by slightly penalizing accuracy. In Section 8, we discuss the support of other sliding window models. Performance evaluation results, based on real life as well as synthetic data sets, are given in Section 9, whereas Section 10 concludes the work.

2 Related Work

Preferences have been used in disciplines such as Game Theory (e.g., Pareto optimality [24]), Computational Geometry (e.g., maximal vectors [16]), Multicriteria Optimization [28], just to name a few. Recently, they have been applied to databases as well [12], [14]. Preference-based queries received considerable attention, due to their usefulness in selecting the most "desirable" objects, especially when the selection criteria are contradictory. In databases, preference-based selections usually take the form of either top-k [8] or skyline queries [4].

Fagin's pioneering work [7], [8] inspired many of the subsequent research efforts [20] to design algorithms that operate under the assumption that the values in each dimension can be provided in sorted order. An extensive survey of top-k query processing techniques in relational databases can be found in [13]. Recently, the reverse top-k query [33] has been proposed and studied in relational databases. The literature is also rich in algorithms to support skyline query processing [4], [6], [25]. In [25], an efficient branch-and-bound skyline query processing scheme has been proposed, which utilizes the R-tree index [10] and shows significant performance improvements over previously proposed methods.

The main drawback of skylines is that the number of points comprising the result is not bounded. This means that a skyline query may return an excessive number of points, generating a cumbersome result. Toward alleviating this problem, several techniques have been designed. For example, k-dominant skylines, proposed in [5], relax the definition of dominance, to allow some points to be dominated, thus, reducing the cardinality of the skyline set. In [29], a variation of skyline queries, the *representative skyline*, has been studied. Another technique has been proposed in [18] for selecting skyline points according to their domination capabilities. More specifically, the algorithm selects a subset of the skyline points aiming at maximizing the total number of dominated tuples. However, this method is NP-hard for high-dimensional spaces, and therefore, approximation algorithms are required toward fast computation.

Ranking of objects according to their domination power has been addressed in [17], [26], [36], [37]. In [36], [37], the authors propose efficient algorithms to determine the top-k dominating points by using an aggregate R-tree index. In [26], a method is studied to rank and cluster multidimensional points according to their domination power. In [17], the authors study 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. Incremental evaluation of top-k dominating queries is examined in [15]. However, the proposed method does not take into account stream characteristics such as the expiration time of points, and therefore, is inappropriate to handle high speed streams. The concept of dominance score has been also used in [27] to rank web services based on multiple criteria.

The common characteristic of all the aforementioned approaches is that they offer solutions to process ad hoc queries, whereas they do not provide mechanisms for continuous query processing over multidimensional data streams. Therefore, for each insertion/deletion of a point, the ranks of active points are computed from scratch. Toward continuous query evaluation, streaming algorithms for skyline [30] and top-k [23] query processing have been proposed. In these works, data form a streaming sequence of points, where each point is characterized by its arrival and expiration time instances, and a sliding window is used to define the set of active points (i.e., the most recently arrived points). More specifically, in [30], an incremental algorithm for continuous skyline queries has been proposed, based on the expiration time of points and the R-tree index. Additionally, continuous evaluation of top-k queries has been proposed in [23]. The proposed algorithm transforms a continuous top-k to a k-skyband query [25], based on the observation that the records that appear in some result of top-k are the ones that belong to the k-skyband in the score-time space. These algorithms are inspired by previously proposed techniques for continuous multidimensional query processing, such as the ones studied in [22].

The efficiency of the majority of the aforementioned techniques relies heavily on the use of pruning, i.e., the elimination of points toward faster processing. Unfortunately, the pruning techniques applied in those algorithms cannot be applied in the case of continuous top-k dominating queries. Due to the challenging nature of the problem, fast pruning-free solutions are required to guarantee accuracy, bookkeeping structures should be efficient to support high update rates and the proposed algorithms should be easily adapted to different sliding window schemes, to cover a broad range of applications. In the sequel, we develop our proposal toward efficient processing of continuous top-k dominating queries over multidimensional data streams, by studying both exact and approximate solutions.

3 Background

Let $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ be a set of d-dimensional points. We use the symbol $p_{i,j}$ to denote the value of the jth dimension of the ith point. A point p_x dominates p_y ($p_x \prec p_y$), if $\forall j \in \{1, \ldots d\}$, $p_{x,j} \le p_{y,j}$ and $\exists m \in \{1, \ldots d\}$ such that $p_{x,m} < p_{y,m}$. The number of points dominated by p_i is denoted as $score(p_i) : score(p_i) = |\{p_j \in \mathcal{P}, p_i \prec p_j\}|$. The result of a top-$k$ dominating query is denoted by $TOPK$ and contains the $k = |TOPK|$ points with the best (highest) scores. The arrival and expiration time of a point p_i is denoted as $p_i.arr$ and $p_i.exp$, respectively. For convenience,

TABLE 1
Frequently Used Symbols

Symbol	Interpretation
$\mathcal{P}$	set of active points
n	number of active points ($n = \|\mathcal{P}\|$)
d	number of dimensions
d_j	the j-th dimension, j = 1, ..., d
p_i	the i-th point, i = 1, ..., n
$p_{i,j}$	the value of the i-th point in the j-th dimension
$p_i.arr$	the arrival time of point p_i
$p_i.exp$	the expiration time of point p_i
$score(p_i)$	the number of points dominated by p_i
$TOPK$	the set of k points with the best scores
k	number of points in the result ($k = \|TOPK\|$)
now	the current time instance
$score_j$	the j-th best $score$ value of the top-k points
exp_j	the j-th min expiration time of points in $TOPK$
g	number of cells per dimension (total #cells = g^d)
c_i	the i-th grid cell
$\Phi(p_i)$	points dominating p_i and expiring later than p_i

the jth best score is written as $score_j$, whereas by exp_j we denote the jth smallest expiration time for points contained in $TOPK$. The current time instance is monitored by the variable now. Table 1 summarizes the basic symbols used. In this work, we focus on the following problem:

Problem Definition. Given *a dynamic data stream containing n active points* $p_1, \ldots, p_n$*, where* $p_i \in \mathbb{R}^d$ *and n is the sliding window size,* monitor *the k points with the highest domination score* continuously.

There exist two basic sliding window types. In a *count-based* sliding window, the number of active points remains constant, and if r new points arrive the r oldest expire. In a *time-based* sliding window, the number of active points may not be constant. The expiration time of a point does not depend on the arrival or expiration of other points. The set of active points is composed of all points arrived the last T time instances. For presentation simplicity, we adopt the count-based sliding window, where the arrival of a new point triggers the expiration of the oldest one. In this scheme, time progresses in every update, whereas the arrival and expiration time instances of a point p_i satisfy the formula: $p_i.exp = p_i.arr + n$. Later, we discuss how our algorithms can handle other sliding window schemes.

A grid-based indexing scheme is used for bookkeeping purposes and for maintaining simplicity in the presentation of the algorithms. We note, however, that our algorithms do not strictly depend on the indexing structure, and therefore, other access methods could be used by performing the necessary modifications. It has been observed that this simple index structure has excellent performance in highly dynamic environments [23], [32], [35]. Each grid cell contains the IDs of the points contained in this cell. Cell c_i dominates completely all cells residing in the upper right region with respect to the upper right corner of c_i. For example, cell c_6 in Fig. 2 dominates cells c_{11}, c_{12}, c_{15}, and c_{16}. These cells are called *fully dominated cells* or simply *dominated cells*. On the other hand, cells c_6, c_7, c_8, c_{10}, and c_{14} may contain points that are dominated by a point in c_6. These cells are called *partially dominated cells*. In Fig. 2, the cells that are fully dominated by c_6 are dark shaded, whereas these that are partially dominated by c_6 are light shaded.

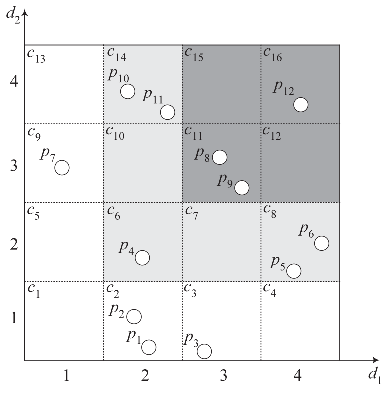


Fig. 2. Fully and partially dominated cells.

In addition to fast bookkeeping, the grid is used to compute the domination score $score(p_i)$ of a point p_i. First, we determine the cell c_j containing p_i. Note that this operation is very fast and it only takes constant time. To compute $score(p_i)$, it is necessary to count the number of points dominated by p_i. These points are contained in the cells that are fully or partially dominated by the cell hosting p_i. For instance, in Fig. 2, p_4 dominates p_6 and p_{11} which reside in partially dominated cells. Moreover, there are three points (p_8, p_9, and p_{12}) in the fully dominated cells. Therefore, the total score of p_4 is $score(p_4) = 2 + 3 = 5$. This process is called *exact score computation* and it is the most time-consuming task.

The naïve approach to evaluate a top-k dominating query continuously is to perform all domination checks among points. More specifically, for a new point p_x, its score $score(p_x)$ is computed by counting the number of points dominated by p_x. Moreover, the score of a point p_y, $y \neq x$ should be increased if p_y dominates p_x. When a point expires, the scores of other points need to be updated. We call this simple algorithm *B*rute-*F*orce *A*lgorithm (BFA). It is expected that BFA will invoke a large number ($O(n)$) of domination checks between points.

4 Event-Based Processing

In this section, we propose an *event-based* algorithm (EVA) which uses event scheduling and rescheduling toward avoiding the examination of points for inclusion in $TOPK$. Let p_i be a point that is not part of $TOPK$ and therefore, $score(p_i) < score_k$ (recall that $score_k$ is the kth best score). In each update, the value of $score_k$ can be reduced at most by 1 and the value of $score(p_i)$ can be increased at most by 1. Therefore, p_i cannot be in $TOPK$ in less than $\lceil (score_k - score(p_i))/2 \rceil$ time instances, unless a top-k dominating point expires during this period. Thus, we can determine a *safe interval* of time as suggested in the following lemma:

Lemma 1. *Given the minimum expiration time* exp_1 *of the top-k dominating points and the current time instance now, a point* p_i *cannot be part of* $TOPK$ *in less than* $SI_1(p_i)$ *time instances, where*

$$SI_1(p_i) = \min\{\lceil (score_k - score(p_i)/2 \rceil, exp_1 - now\}. \quad (1)$$

Proof. There are two possible cases in which p_i can be part of $TOPK$ in each update: 1) if a point in $TOPK$ expires or 2) if the score of p_i becomes larger than or equal to $score_k$. The first case is covered by the second part of (1), since exp_1 is the minimum time in which a top-k point expires. For the second case, we assume the worst case scenario, in which $score_k$ decreases by 1 and $score(p_i)$ increases by 1 in each update (the kth best point dominates the expired point and does not dominate the new point whereas for p_i the opposite is true). Therefore, the difference between their scores is reduced at most by 2 in each update. Thus, at least for the next $\lceil (score_k - score(p_i))/2 \rceil - 1$ time instances, $score(p_i)$ is less than $score_k$. In summary, there are only two cases in which a point can be part of $TOPK$ and both of them are covered by (1). □

The safe interval $SI_1(p_i)$ serves as a lower bound, indicating that during this interval, p_i is impossible to enter $TOPK$ and thus, it is ignored. If now is the current time, p_i will be examined again as a possible candidate for $TOPK$ at time $\min(\lceil (score_k - score(p_i))/2 \rceil + now, exp_1)$. An event e_i is associated with every point $p_i \notin TOPK$ and contains the following data:

- the *event processing time*, denoted as $e_i.ept$, indicating the time that p_i should be examined as a top-k candidate, $e_i.ept = \min(\lceil (score_k - score(p_i))/2 \rceil + now, exp_1)$,
- the *event generation time*, denoted as $e_i.egt$, storing the time instance that the event was generated, and
- the score of p_i at time $e_i.egt$, denoted as $e_i.score$.

Each event e_i is thus represented as a triplet $<ept, egt, score>$. All scheduled events are organized by a priority queue, using the ept field for prioritization. The head of the queue contains the event that will be processed next, having the minimum ept value. This event will be processed if the current time equals the event processing time. For instance, if e_i is on the head, this event will be processed when $e_i.ept = now$. When the event e_i is processed, the following actions are taken: 1) the exact score of p_i is computed, 2) a new event is scheduled for p_i by computing a new value for ept.

Assume for the time being that there is no indexing scheme available. Evidently, the aforementioned approach will be more efficient than BFA, if $SI_1(p_i)$ is long enough so that the cost of BFA during this period will be higher than that of an exact score computation. Each exact score computation costs n domination checks among points in the worst case. In BFA, each point p_i requires two domination checks per update; one for the incoming point and one for the expiring one. Therefore, the number of domination checks performed by BFA for the duration of the safe interval is $2(e_i.ept - e_i.egt)$. Consequently, BFA performs more domination checks when $2(e_i.ept - e_i.egt) \geq n \Rightarrow (e_i.ept - e_i.egt) \geq n/2$. This means that larger safe intervals are favorable since they lead to fewer exact score computations and therefore, less computational cost. However, safe intervals with duration more than $n/2$ are difficult to be produced, especially for large n and small k. For this reason, in the sequel we study two methods used by EVA to reduce the number of exact score computations. The first one focuses on bounding the score of existing points, whereas the second estimates the score of a newly arrived point.

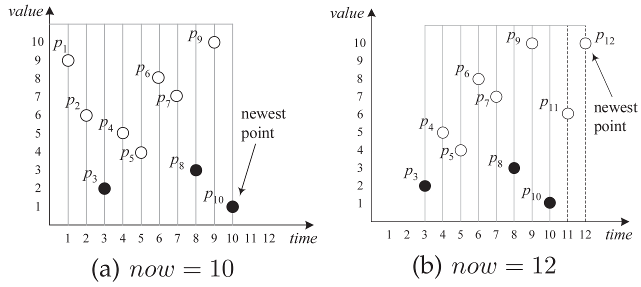


Fig. 3. Event time computation example.

4.1 Upper-Bounding Scores of Existing Points

The event processing time $e_i.ept$ of an event e_i is computed taking into account the worst case scenario, where $score_k$ is reduced by 1 in each update, meaning that the kth point of $TOPK$ dominates all the outgoing points and none of the incoming points that arrived between the time $e_i.egt$ and $e_i.ept$.

When an event is processed, we can reschedule the event, avoiding computing the score of the associated point, if we keep up-to-date the scores of the top-k dominating points. Notice that the cost to update k scores is affordable since $k \ll n$. To compute the new event processing time, the score of the point is required which is not known. However, it is safe to use an upper bound: the score p_i is less than or equal to $e_i.score + e_i.ept - e_i.egt$. We use this bound in place of $score(p_i)$ in (1) to compute the new safe interval and thus, to schedule a new event for p_i.

Fig. 3 shows a sliding window of length $n = 10$ with 1-dimensional points, where the horizontal axis shows time instances. For convenience, we assume that point p_i arrived at the ith time instance. Fig. 3a shows the set of active points when $now = 10$. For $k = 3$, the top-3 dominating points are p_{10}, p_3, and p_8 (black dots) and their corresponding scores are 9, 8, and 7, respectively. The minimum expiration time of the top-3 dominating points is $exp_1 = 3 + 10 = 13$ (p_3 will expire first) and the third best score is $score_3 = 7$ (due to p_8). We illustrate the computation of the event e_7 of p_7. Currently, we have that $score(p_7) = 3$ since p_7 dominates p_1, p_6, and p_9. By substituting the values in (1), we have that $e_7.ept = \min(\lceil (7-3)/2 \rceil + 10, 13) = 12$, $e_7.egt = 10$, and $e_7.score = 3$. This means that p_7 will be examined again as a possible candidate for $TOPK$ when $now = 12$, i.e., in the next two updates. Assume now that two updates occur and thus, $now = 12$ as shown in Fig. 3b. The two oldest points (p_1 and p_2) have been expired and the set of active points is $\{p_3, \ldots, p_{12}\}$. Since $e_7.ept = now$, point p_7 should be examined. We recompute the event processing time of point p_7. The upper bound score estimation is $score(p_7) \leq e_7.score + e_7.ept - e_7.egt = 3 + 12 - 10 = 5$. We use this upper bound to recompute the event time, thus avoiding the exact score computation for $score(p_7)$. The value of $score_3$ is still 7 and therefore, $e_7.ept = \min(\lceil (7-5)/2 \rceil + 12, 13) = 13$, $e_7.egt = 12$, and $e_7.score = 5$, meaning that point p_7 will be examined again in the subsequent update where $now = 13$.

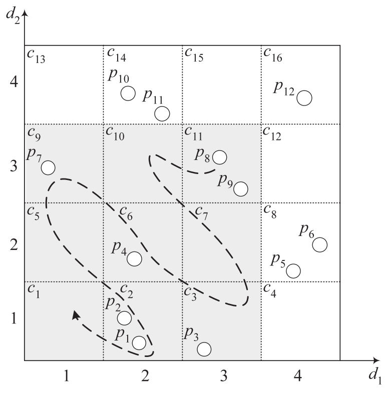


Fig. 4. Cell visiting order.

4.2 Upper Bounding Scores of Incoming Points

The computation of the event processing time $e_i.ept$ requires the score of p_i or at least an upper bound. However, in the streaming scenario, new points continuously arrive and there is no information about their scores. To avoid computing scores from scratch, we try to find an upper bound of them. Let p_x be an incoming point. The key idea is to determine a point p_r with the following properties: 1) it dominates the incoming point p_x, and 2) it is not part of $TOPK$. We denote this point as a *reference* point. If such a point exists, then we can compute an upper bound for $score(p_x)$, and use it in (1) to calculate the event time of p_x. If now is the current time instance then $score(p_r) \leq e_r.score + now - e_r.egt$. Since, p_r dominates p_x then $score(p_x) \leq score(p_r) - 1 \Rightarrow score(p_x) \leq e_r.score + now - e_r.egt - 1$. If there is no point satisfying these properties, then an exact score computation for p_x is performed.

To locate a point that dominates the new point p_x, we search the grid cells that partially or fully dominate the cell c_j containing p_x. These cells lie at the lower left region with respect to the upper right corner of c_j. A quick way to determine a point that dominates p_x and has the lowest score, is to use a cell visiting order as the one shown in Fig. 4. Assume that a new point has been inserted in cell c_{11}. To determine the cells to be visited next, we use the locations of the cells in each dimension. The *sum of cell coordinates* (scc), specifies the search order. Cells with large scc values are examined first. After visiting c_{11} ($scc(c_{11}) = 6$), the next cells in order are c_7 and c_{10} ($scc(c_7) = scc(c_{10}) = 5$). Ties are broken arbitrarily. Next, we visit c_3, c_6, and c_9 ($scc(c_3) = scc(c_6) = scc(c_9) = 4$). The search continues until either a convenient point p_r is found, or cell c_1 is reached.

4.3 Outline of EVA

The pseudocode of EVA is depicted in Fig. 5. For each update, a sequence of operations is applied. First, the bookkeeping structure is updated (Line 1), meaning that the expiring point is deleted and the new one is inserted. Then, the scores of the top-k dominating points are updated (Line 1).

The incoming point p_x is processed to determine if it should be part of $TOPK$ (Lines 2-8). The procedure FindReferencePoint() is called, which tries to locate a point p_r such that: 1) it dominates p_x and 2) it is not part of $TOPK$. If such a point does not exist or the upper bound of the score is larger than or equal to $score_k$, then the score

```
Algorithm EVA ($p_x$, $now$)
    $p_x$: the incoming point
    $now$: the current time instance

1.  update index; update scores of points in $TOPK$;
2.  $e_r.score = 0$; $e_r.egt = now$;
3.  $p_r \leftarrow$ FindReferencePoint($p_x$); $e_r \leftarrow$ event of $p_r$;
4.  $score(p_x) \leftarrow e_r.score + now - e_r.egt - 1$;
5.  if ($score(p_x) \geq score_k$) then
6.      compute $score(p_x)$ from scratch;
7.      if ($score(p_x) \geq score_k$) then insert $p_x$ in $TOPK$;
8.  if ($p_x \notin TOPK$) then ScheduleEvent($x$,$score(p_x)$,$now$);
9.  $e_i \leftarrow EventQueue$.RemoveTop(); // get the first event
10. while ($e_i.ept = now$);
11.     if (number of top-$k$ points $\geq k$) then
12.         $score \leftarrow e_i.score + e_i.ept - e_i.egt$;
13.         ScheduleEvent($i$,$score$,$now$);
14.     if ($e_i$ is not rescheduled) then
15.         compute $score(p_i)$ from scratch;
16.         if ($score(p_i) \geq score_k$) then insert $p_i$ in $TOPK$;
17.         else ScheduleEvent($i$,$score(p_i)$,$now$);
18.     $e_i \leftarrow EventQueue$.RemoveTop() // get the next event

19. function ScheduleEvent($j$, $score$, $now$)
20.     $exp_1 \leftarrow$ minimum expiration time of points in $TOPK$;
21.     $ept \leftarrow \min(\lceil (score_k - score)/2 \rceil + now), exp_1)$;
22.     if ($ept \geq now$) then
23.         $e_j.ept \leftarrow ept$; $e_j.egt \leftarrow now$;
24.         $e_j.score \leftarrow score$;
25.         $EventQueue$.Insert($e_j$);
```

Fig. 5. Outline of EVA.

of the p_x is computed from scratch. If $score(p_x)$ is greater than $score_k$, p_x is inserted in $TOPK$. Otherwise, an event for p_x is generated using the procedure ScheduleEvent() (Lines 19-26). This procedure takes three parameters: the id j of the point, its $score$ (exact or an upper bound) and the current time instance now. First, it computes the event processing time and then, if this time is greater or equal to now, it inserts the event into the priority queue. Notice that, the event processing time is less than now, if the parameter $score$ is larger than $score_k$. Thus, in Line 8, an event is always generated, since either the upper bound (Line 5) or the exact score (Line 7) is less than $score_k$.

Finally, all events with event processing time equal to now are processed (Lines 9-18). For an event e_i, we try to recompute its $e_i.ept$ value by using the upper bound of $score(p_i)$ (Lines 12-13). If the event is reinserted into the event priority queue, we consider the next event; otherwise, if the upper bound estimation is poor, it is possible the computed event time to be less than now. In such a case, we proceed with the exact score computation of $score(p_i)$ and either p_i is inserted in the top-k, or the event time is recomputed based on the exact score of p_i (Lines 14-17). Line 11 controls the expiration of a top-k dominating point. If a top-k point expires in now, $score_k$ is not updated and therefore, we should not try to compute event times. In this case, we set $score_k$ to -1, to force the insertion of another point in $TOPK$. The score of the point of the first examined event is computed and the point is inserted in $TOPK$. Next, we try to recompute the event time of the remaining events.

5 The Advanced Algorithm (ADA)

EVA has two important limitations. The first is that all points that are not part of $TOPK$ should be examined at the expiration time of a top-k dominating point. Moreover, the event time computation gives a large number of events and only a small percentage of them will cause a change in $TOPK$. The second is that it is possible that many points have a score close to $score_k$, resulting in consecutive exact score computations. In the sequel, we discuss two significant optimizations toward alleviating these drawbacks, leading to the design of algorithm *AD*vanced *A*lgorithm.

5.1 Advanced Event Time Computation

Let $\Phi(p_i)$ denote the set of points that dominate p_i and expire later than p_i, i.e., $\Phi(p_i) = \{p_j \in \mathcal{P}, p_j \prec p_i \wedge p_j.exp > p_i.exp\}$. Moreover, we denote by $\Phi^+(p_i)$ the subset of $\Phi(p_i)$ contained in $TOPK$, and by $\Phi^-(p_i)$ the subset of $\Phi(p_i)$ not contained in $TOPK$. Consequently, $\Phi(p_i) = \Phi^+(p_i) \cup \Phi^-(p_i)$ and $\Phi^+(p_i) \cap \Phi^-(p_i) = \emptyset$. The key observation for a more efficient event time computation lies on the fact that if $|\Phi^-(p_i)| > 0$, then p_i has a chance to be part of $TOPK$ only if these points will be included in $TOPK$ in the future. Assume that p_i is dominated by r points which are not part of $TOPK$ and all of them expire later than p_i (i.e., $|\Phi^-(p_i)| = r$). Then, the following holds:

Lemma 2. *Given the rth minimum expiration time of the top-k dominating points, denoted as* exp_r $(0 \leq r < k)$ *and the current time instance now, a point* p_i *with score* $score(p_i)$ *cannot be part of* $TOPK$ *in less than* $SI_2(p_i)$ *time instances, where*

$$SI_2(p_i) = \min\{\lceil (score_{k-r} - score(p_i))/2 \rceil, exp_{r+1} - now\}. \tag{2}$$

Proof. It suffices to prove that p_i can be part of $TOPK$ if at least $r+1$ top-k points expire or $score(p_i) \geq score_{k-r}$ (the $(k-r)$th best score). Since there are r points that dominate p_i and expire later than p_i, these points have always higher score than p_i during its lifetime and therefore, p_i can be part of $TOPK$ only after these points have been inserted in $TOPK$. If less than $r+1$ top-k points expire, all the available positions in $TOPK$ will be covered by the points contained in $\Phi^-(p_i)$. Similarly, $score(p_i)$ should be larger than $score_{k-r}$, otherwise the r points, which have higher score than $score(p_i)$, will be inserted in $TOPK$ and therefore, the kth score will be increased and will be larger than $score(p_i)$. □

It is not hard to show that the safe interval computed by Lemma 1 is less than or equal to that computed by Lemma 2, i.e., $SI_1(p_i) \leq SI_2(p_i)$. Equation (2) corresponds to (1), if $r = 0$. Otherwise, (2) computes always larger safe intervals. Therefore, by using $SI_2(p_i)$ fewer events are generated and consequently, the computational cost decreases. For instance, in Fig. 3 assume that $now = 10$, whereas $TOPK = \{p_{10}, p_3, p_8\}$ with scores 9, 8, and 7, respectively. According to Lemma 1, the event time of p_4 is $e_4.ept = \min(\lceil (7-5)/2 \rceil + 10, 13) = 11$. Since p_5 dominates p_4 and expires later than p_4, it holds that $|\Phi^-(p_4)| = 1$, which means that we can use the $(k-1)$th score $(score_{k-1})$ and the second minimum expiration time of the top-k dominating points (exp_2). Thus, according to Lemma 2, $e_4.ept = \min(\lceil (8-5)/2 \rceil + 10, 18) = 12$.

A naïve approach to maintain $|\Phi^-(p_i)|$ is to use a counter for each p_i and to update these counters for each arrival.

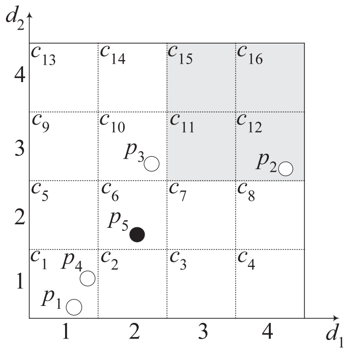


Fig. 6. Advanced event time computation.

More specifically, if the new point dominates p_i then we increase the counter of p_i by 1. Since in the worst case all counters may require an update, it is expected that this approach will be inefficient. Thus, we use a different method which maintains a counter for each point, $p_i.counter$, and a counter for each cell, $c_j.counter$. The difference $c_j.counter - p_i.counter$ is used as a lower bound for $|\Phi^-(p_i)|$. For instance, assume the 2-dimensional grid of Fig. 6 and let $n = 4$ and $k = 1$; thus, the answer of the query is p_1. A new point p_5 (black dot) arrives and p_1 expires. We increase by 1 all counters of cells fully dominated by the cell containing p_5, i.e., the shaded cells of Fig. 6. The cost of this process is marginal. Moreover, we set the counter of the new point equal to the counter of the cell containing the new point; thus, we have $p_5.counter = c_6.counter$. For this update, the cell counters are updated and the counter of the new point is initialized.

To use the difference $c_j.counter - p_i.counter$ as a lower bound for $|\Phi^-(p_i)|$ for a point p_i, we should update the point counters when there is a change in $TOPK$. In the case where a point p_j is inserted into $TOPK$, we increase by 1 the counters of all points dominated by p_j and expire before p_j, whereas in the case of a top-k expiration, the counters of these points are decreased by 1. In the example, p_4 is inserted into $TOPK$ because p_1 expires. The counters of p_2 and p_3 are increased, since p_4 dominates both of them and expires later than them. Before the update, we have $p_3.counter = 1$, $c_{10}.counter = 2$ and therefore, $c_{10}.counter - p_3.counter = 1$. After the update, $p_3.counter$ increases by 1 (due to the insertion of p_4 in $TOPK$) and thus, $c_{10}.counter - p_3.counter = 0$. In fact, $|\Phi^-(p_3)| = 1$ since p_5 dominates p_3, expires later, and $p_5 \notin TOPK$. The use of a lower bound for $|\Phi^-(p_i)|$ results in a smaller safe interval according to (2), but it will not introduce false positives in the result.

A better lower bound can be determined for $|\Phi^-(p_i)|$ by considering points that partially dominate the cell containing p_i. In the case where an exact score computation is necessary for p_j, we decrease by 1 the counters of all points for which the following hold: 1) they belong to a cell that is partially dominated by the cell containing p_j, 2) p_j dominates these points, and 3) p_j expires later than these points. In our example, assume that we cannot estimate the score of the new point p_5 and therefore, an exact score computation is carried out. We decrease $p_3.counter$ by 1, since p_3 belongs to cell c_{10} which is partially dominated by c_6, p_5 dominates p_3 and p_5 expires later than p_3. Therefore,

$|\Phi^-(p_3)| = 1$. Notice that this process does not impose additional overhead, because it is applied during exact score computations.

5.2 Using Candidate Points

As mentioned before, one drawback of EVA is that if some points exist that their score is close to $score_k$, many exact score computations will be performed. The same holds even if the advanced event computation is applied. To overcome this problem, we propose to continuously evaluate the score of some special points called *candidate points*, whose event processing time is in the near future. The challenge is to maintain candidate points aiming at reducing the number of exact score computations.

To illustrate the idea of candidate points, assume a top-3 dominating query is applied in 1,000 points and $score_3$ is 200. Assume further that there is a point p_i with score 194 that expires after the points of $TOPK$. The event e_i will be processed in 3 time units. However, it is highly probable that in a so small time interval the scores of $TOPK$ and p_i will not change. Therefore, an exact score computation will happen and e_i will be arranged to be processed again in 3 time units. When there are no drastic changes in $TOPK$, this process can continue for a large part of the lifespan of p_i and probably p_i will never be part of $TOPK$.

When an event e_i for point p_i is computed, if $e_i.ept \leq now + \Delta t$ we insert p_i directly into the set of candidate points and we ignore the event; otherwise, we proceed in the normal way. The time threshold Δt is used to decide whether the event processing time is in the "near" future or not. To avoid a large number of candidate points, the parameter $maxcand$ is used to define the maximum allowed number of maintained candidates. In our experiments, Δt is initialized to $n/1{,}000$, and its value is adjusted automatically as follows: it decreases by 1 if the number of candidate points is greater than or equal to $maxcand$ and increases by 1 if exact score computations are taking place. The value of $maxcand$ is set to 1 percent of the sliding window size.

5.3 Evicting Obsolete Events

In top-k dominating queries, we cannot prune points in contrast to skyline and top-k queries, since the domination power of the points depends on the other points. Therefore, the indexing structure used contains all points of the sliding window. On the other hand, events can be pruned, if the event processing time of a point is greater than its expiration time. More specifically, let p_i be a point with an associated event e_i. If $p_i.exp < e_i.ept$, then p_i will expire before the event is processed. Such an event is characterized as *obsolete*. We can reduce the storage requirements if obsolete events are evicted. However, with less events fewer reference points are available, and thus we may affect running time. This is a tradeoff between memory consumption and efficiency, and it is further studied in the performance evaluation.

6 Performance Analysis

In this section, we perform an analysis toward upper bounding the expected number of exact score computations required for a sequence of m updates. Recall, that in our model, each arrival of a new point is immediately followed by an expiration of the oldest one. Following related research in the area [23], we assume that: 1) points are distributed uniformly in the unit d-dimensional hypercube, and 2) there is no correlation between the dimensions.

The cost of an exact score computation depends on the number of points that must be examined. Let g denote the number of cells in each dimension. Therefore, the total number of cells equals g^d. The following lemma gives the expected number of the partially dominated cells for any grid cell.

Lemma 3. *The expected number of points that must be examined during an exact score computation is given by*

$$\frac{n}{2^d} \cdot \left(\left(\frac{g+1}{g} \right)^d - \left(\frac{g-1}{g} \right)^d \right). \tag{3}$$

Proof. By treating each dimension separately, the expected number of partially plus the number of fully dominated cells is given by

$$\overline{FDPD} = \frac{1}{g^d} \cdot \prod_{j=1}^{d} \sum_{i=1}^{g} i = \left(\frac{g+1}{2} \right)^d.$$

Again, by treating each dimension separately, the expected number of fully dominated cells is given by

$$\overline{FD} = \frac{1}{g^d} \cdot \prod_{j=1}^{d} \sum_{i=1}^{g-1} i = \left(\frac{g-1}{2} \right)^d.$$

By subtracting $\overline{FD}$ from $\overline{FDPD}$, we get that the expected number of partially dominated cells is

$$\overline{PD} = \left(\frac{g+1}{2} \right)^d - \left(\frac{g-1}{2} \right)^d.$$

Due to the uniformity assumption, there are on average $\frac{n}{g^d}$ points in each cell, and thus, the result follows. □

In the sequel, we provide an upper bound on the expected number of exact score computations required for m update operations. According to EVA, there are three cases where an exact score computation can happen during an update: 1) an incoming point must enter $TOPK$ or the skyline of the rest of the points (with probability P_a), 2) the expiring point belongs to $TOPK$ (with probability P_e), and 3) an examined event associated with a point p cannot be rescheduled (with probability P_p). In our analysis, we use the result of Godfrey [9], which states that the expected number of skyline points $\overline{sky}(u)$ for a set of u points in d dimensions by assuming independence among the dimensions is $O(ln^{(d-1)}u/(d-1)!)$.

The probability that an incoming point p_x will cause an exact score computation is at most the probability that: 1) p_x will land on a cell occupied by one of the k points of $TOPK$, 2) on a cell containing at least one skyline point of the rest n-k points, or 3) on an empty cell lying in the lower left region of the skyline border. By the uniformity and independence assumption, it follows that the expected number of empty cells is $c/e^{n/c}$, where $c = g^d$ is the total number of cells. The result follows by considering that we have a balls-and-bins game and using the linearity of expectation as shown in

[21]. By using the pessimistic assumption that each $TOPK$ point and each of the $\overline{sky}(n-k)$ points lie in a different cell, it follows that p_x will cause an exact score computation with probability[2] at most

$$P_a \leq \frac{k + \overline{sky}(n-k) + c/e^{n/c}}{c}. \tag{4}$$

When a point expires, with probability P_e it will be one of the k points of $TOPK$. According to the uniformity and independence assumption it holds that

$$P_e = k/n. \tag{5}$$

Moreover, the processing of an event associated with point p will cause an exact score computation, if and only if $score_k$ is reduced in every update occurred during the safe interval $SI_1(p)$. Otherwise, the event will be rescheduled. The value of $score_k$ is reduced during an update if the kth best point dominates the expired point and does not dominate the new point. This probability equals the volume dominated by the kth point multiplied by the remaining volume for each update during $SI_1(p)$. Therefore, we have

$$\begin{aligned} P_p = &\frac{score_k}{n} \cdot \frac{n - score_k}{n} \cdot \frac{score_k - 1}{n} \cdot \frac{n - score_k + 1}{n} \\ &\cdots \frac{score_k - SI_1(p) + 1}{n} \cdot \frac{n - score_k + SI_1(p) - 1}{n} \\ \Rightarrow P_p = &\prod_{j=0}^{SI_1(p)-1} \frac{(score_k - j) \cdot (n - score_k + j)}{n^2}. \end{aligned}$$

To simplify the analysis, we use only the first term of the product. Therefore, the probability P_p, that the processing of an event will cause an exact score computation satisfies the following inequality:

$$P_p \leq \frac{score_k}{n} \cdot \frac{n - score_k}{n}. \tag{6}$$

When an update occurs, an expiration with probability P_e will cause at most $\overline{sky}(n-k)$ exact score computations. Otherwise, with probability P_a one exact score computation will take place. In case where neither a top-k expires or a top-k arrives or the new point belongs to the skyline of the $n-k$ points, an update can cause at most $\overline{sky}(n-k)$ exact score computations with probability P_p. Therefore, we can derive the expected number of exact score computations in the ith update, denoted as $E[\mathcal{X}_i]$, by using (4), (5), and (6). Let $\mathcal{X} = \mathcal{X}_1 + \mathcal{X}_2 + \cdots + \mathcal{X}_m$ be the random variable counting the total number of exact score computations performed in m updates. By the linearity of expectation, we have that $E[\mathcal{X}] = \sum_{0 \leq i \leq m} E[\mathcal{X}_i]$. By substitutions and algebraic manipulations, we get that $E[\mathcal{X}]$ is

$$O\big(m(P_a \cdot (1 - P_e) + P_e \cdot S + (1 - P_a) \cdot (1 - P_e) \cdot P_p \cdot S)\big),$$

where $S = \overline{sky}(n-k)$. The above bound is quite pessimistic, since several worst case assumptions have been performed. For example, we have assumed that when a new point enters the skyline of the n-k points, then $\overline{sky}(n-k)$ exact score computations will be performed, which is an overestimation. In addition, we assumed that all empty cells estimated by the balls-and-bins game are located in the lower left corner, which again is an upper bound. We avoided the use of simplistic arguments aiming at an analysis with theoretic basis. In practice, as we have validated through experimentation, the number of exact score computations performed is far less. However, this is the first step to provide a closed-form formula to estimate the cost of the algorithms.

2. Evidently, this holds in the case that the parameter c assumes a convenient value to ensure that P_a is less than or equal to 1.

7 Approximate Solutions

So far, our study has centered on *exact* algorithms, providing 100 percent correct and up-to-date answers to a top-k dominating query. However, in many modern applications (e.g., decision support, information retrieval), exact answers are, typically, not needed to draw useful conclusions; instead, *approximate* results are often sufficient [3], [31]. The effectiveness of an approximate solution is measured by considering the *accuracy* of the result, which corresponds to the fraction of the correct top-k dominating points returned by the algorithm over the actual number of top-k dominating points. In the sequel, we study two approximate solutions: 1) a randomized approach based on probabilistic bounds and 2) a more aggressive approach without probabilistic guarantees.

7.1 Approximation with Probabilistic Guarantees

The exact algorithms calculate the event time of a point p_i based either on the exact score or the maximum score. To evaluate the query accurately, the maximum possible score is computed by using the worst case scenario, in which the score of the kth dominating point is reduced in each update, whereas the score of p_i increases. This way, a safe upper bound is generated. However, the exact score of p_i rarely (if not ever) approaches the maximum value. By using an approximation, a more realistic value (closer to the exact score) may be computed, speeding up query execution and, on the other hand, possibly sacrificing the accuracy of the result.

From elementary probability theory, it holds that the sum of squared errors around an arbitrary value is minimized if this value is the true mean. The key idea is to maintain a random sample [34], [2], [38] of points in a cell and compute their mean score. This value is used as an approximation for the score of any other point of this particular cell. The sample size should be large enough so as to guarantee that the estimated score will not be very far from the true mean. To determine a convenient sample size, we use the Hoeffding inequalities [11]. Assume s independent random variables $\mathcal{Z}_1, \ldots, \mathcal{Z}_s$ such that $\alpha \leq \mathcal{Z}_i \leq \beta$, $i = 1, \ldots, s$. Let $\mathcal{Z} = (1/s)\sum_{i=1}^{s} \mathcal{Z}_i$ and $\mu = E[\mathcal{Z}]$ (the expectation of $\mathcal{Z}$). For any $\xi > 0$, the one-sided and two-sided forms of the Hoeffding inequality have as follows:

$$Prob\{\mathcal{Z} - \mu \leq \xi\} \geq 1 - e^{-2s\xi^2/(\beta-\alpha)^2} \tag{7}$$

$$Prob\{|\mathcal{Z} - \mu| \leq \xi\} \geq 1 - 2e^{-2s\xi^2/(\beta-\alpha)^2}. \tag{8}$$

If $\mathcal{Z} \geq \mu$, then the mean value is overestimated, meaning that no false negatives appear. On the other hand, if $\mathcal{Z} < \mu$ then some points may be missed, because $\mathcal{Z}$ cannot be used as an upper bound. Therefore, we try to bound the error selecting an appropriate value for s to increase the probability that $\mu - \mathcal{Z} \leq \xi$. From (7) and (8), it is not hard to deduce that

$$Prob\{\mu - \mathcal{Z} \leq \xi\} \geq 1 - e^{-2s\xi^2/(\beta-\alpha)^2}. \tag{9}$$

By setting $\delta = Prob\{\mu - \mathcal{Z} \leq \xi\}$ and solving for s, we get an expression that gives the minimum number of samples required to have a maximum error ξ with probability at least δ

$$s = (\beta - \alpha)^2 \cdot \frac{\ln\left(\frac{1}{1-\delta}\right)}{2\xi^2}. \tag{10}$$

In our case, the bound is used to compute the number of samples that must be maintained for each cell c_j, in order to monitor the average score in c_j. The score of an arbitrarily selected point lies between 0 and $n-1$. However, if samples are taken from a specific cell c_j then $minscore(c_j) \leq score(p_i) \leq maxscore(c_j)$. We refer back to Fig. 6 to illustrate the idea. For any point p_i hosted in cell c_1, it holds that $3 \leq score(p_i) \leq 5$. The value of $maxscore$ is determined by counting the number of points contained in the partially and fully dominated cells, whereas the value of $minscore$ equals the number of points contained in the fully dominated cells. Therefore, the number of samples is determined by substituting in (10) the appropriate values for ξ, δ, and setting $\alpha = minscore(c_j)$, $\beta = maxscore(c_j)$. The key observation is that if ξ may be expressed as a percentage of the score range, rather than an absolute value. Formally, by setting $\xi = \varepsilon(\beta - \alpha)$ we have that

$$s = \frac{\ln\left(\frac{1}{1-\delta}\right)}{2\varepsilon^2}. \tag{11}$$

We argue, that it is neither practical nor meaningful to maintain a sample for every cell because: 1) the advanced event computation may be sufficient to exclude some points and therefore, a better estimation is not necessary for these points and 2) a cell may be dominated by another cell for which we already maintain samples. In addition, a large number of sample points may lead to increased costs because the exact scores of the sample points must be monitored during updates. Thus, we maintain samples only for a small number of cells, determined using the maximum possible score of a point in a cell. We use again the parameter Δt, which determines if the event is in the near future or not. The following heuristic is applied: if $maxscore(c_j) > score_k - \Delta t$ and c_j is not dominated by another cell with a sample, then we keep a sample for c_j.

The above technique is combined with ADA, which is the best exact algorithm as will be demonstrated in Section 9. In addition to the estimation performed by ADA for the score of a point p_i, a second estimation is performed using the sample points, provided that a sample exists for the cell containing p_i. Between the two estimated values, the minimum is chosen. The new algorithm is termed *A*pproximate *H*oeffding *B*ound *A*lgorithm (AHBA) and it has the nice property that the error introduced in the score estimation is controllable.

7.2 "Quick-and-Dirty" Approximation

Algorithms ADA and AHBA try to reduce the number of events and the number of exact score computations assuming, pessimistically, that all points have a possibility to become part of the result. However, by assuming that the data distribution does not change rapidly, it is possible to ignore some points by excluding them from the event generation mechanism.

If the score of a point is low, we expect that it will remain low during its lifetime (by assuming that data distribution does not change significantly). Let $minscore(c_j)$ denote the smallest possible score of all points contained in cell c_j. All points belonging to c_j are excluded from event processing if $score_1 - score_k < score_k - minscore(c_j)$. This algorithm, denoted as *A*pproximate *M*inimum *S*core *A*lgorithm (AMSA), uses an aggressive heuristic-based technique to speedup processing, but it does not provide any guarantees regarding accuracy. Evidently, when the data distribution changes rapidly then this heuristic is not appropriate and EVA, ADA, or AHBA should be used instead.

8 Alternative Sliding Window Schemes

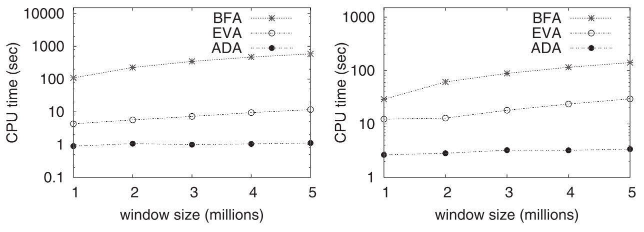
In the beginning of our exploration, we considered that a count-based sliding window is used and in particular, each update involves one arrival of a new point and one expiration of the oldest point. In this section, we discuss briefly how other sliding window models are supported by our algorithms. More specifically, we discuss how Lemmas 1 and 2 adapt to these cases. All the other issues (e.g., event time computation) are addressed in a straightforward manner.

1. **Time-based window with one arrival and one expiration in each update:** Since we have one insertion in each time instance, we also have one expiration in each time instance. Thus, Lemmas 1 and 2 remain unchanged.
2. **Time-based window with multiple arrivals and expirations:** In general, the number of arrivals/expirations varies for each time instance. We compute the minimum number of updates that can occur without p_i being part of $TOPK$. Lemmas 1 and 2 are the same, but their first part measure the number of updates instead of time instances. Therefore, Lemma 1 states that point p_i cannot be part of $TOPK$ in less than $\lceil((score_k - score(p_i))/2)\rceil$ updates or $exp_1 - now$ time instances. Lemma 2 is handled similarly.
3. **Count-based window with multiple arrivals and expirations:** Both Lemmas 1 and 2 give the minimum number of updates required instead of the minimum time instances. Therefore, we need to express $exp_1 - now$ as the number of updates required for the first expiration of a point of $TOPK$. Assume that exp_1 corresponds to the expiration time of p_i, i.e., $p_i.exp = exp_1$. We can use the index i to denote the update (e.g., i is an id of the ith update). Let p_j be the last inserted point. Then, $exp_1 = p_i.exp = i + n$, where n is the size of the window and $now = j$. Again, Lemma 2 is handled similarly.

The previous observations show that the changes required in the proposed algorithms to support other sliding window models are marginal.

9 Performance Evaluation Study

All algorithms have been implemented in C++ and the experiments have been conducted on a Pentium@3.0 GHz



(a) IND (b) ANT

Fig. 7. Running time versus window size (number of points).

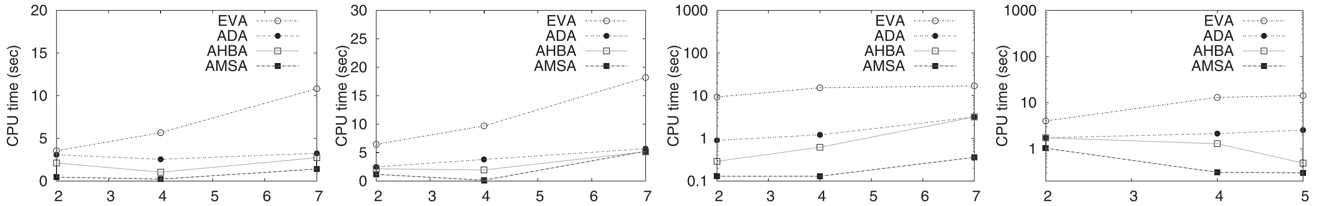
Win XP machine, with 1 GB of RAM. The synthetic data sets are the independent (IND) and anticorrelated (ANT), generated by using the process described in [4]. Additionally, two real-life data sets have been used: Forest Cover (FC) (http://kdd.ics.uci.edu) containing 581,012 records with attributes such as elevation, slope, horizontal distance to hydrology, etc., and Zillow (ZIL) (http://www.zillow.com) containing 1,252,208 real estate records with five correlated attributes.

The computational cost is expressed by the running time per 1,000 updates (arrivals/expirations). The update operations of the startup phase (first n arrivals) is not counted in the measurements. The default values for the parameters, if not otherwise specified, are: $n =$ 2,000,000, $d = 4$ and $k = 256$. The maximum number of candidate points is set to 1 percent of the window size and the parameter Δt is set to $n/1{,}000$. The error ε of the Hoeffding bound is set to 10 percent of the range of a cell, whereas a 90 percent confidence is used, i.e., $\delta = 0.9$.

First, we study the performance of the algorithms with respect to the number of active points (window size). Fig. 7 depicts the CPU time for IND and ANT data sets, by varying the window size from 1 to 5 millions. The y-axis is illustrated in logarithmic scale. It is evident, that BFA is not appropriate for the streaming scenario and therefore it is omitted from subsequent experiments. ADA outperforms EVA, because ADA generates fewer events and performs fewer exact score computations, resulting in better performance. As expected, the difference between the two methods is more significant in the ANT data set, since the difference in score values of the points do not vary significantly. ADA overcomes this problem by computing larger event times due to the advanced event computation and mainly due to the use of candidate points.

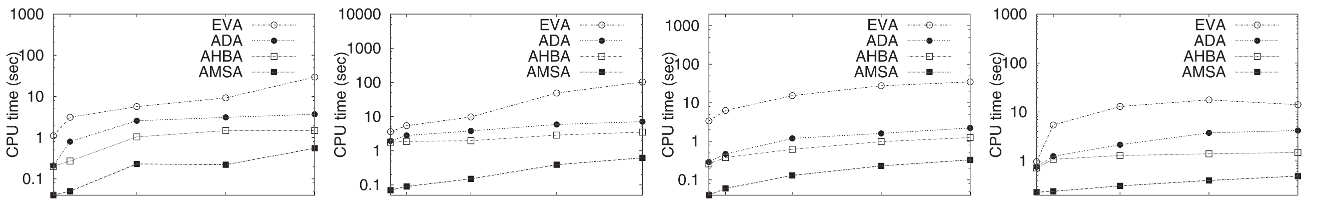
Next, we study the performance with respect to dimensionality. The corresponding results are given in Fig. 8. For IND and ANT data sets, the window size is set to 2 millions, for FC is set to 500,000 and for ZIL is set to 1 million. Again, we observe that ADA is significantly more efficient than EVA due to the reason we mentioned previously. As the dimensionality increases, the scores of the points become lower since the probability that a point dominates another is significantly reduced. Therefore, EVA produces more events than ADA, and more exact score computations are involved. As expected, AHBA and AMSA are always more efficient than EVA and ADA. In particular, AMSA shows the best performance since it does not use samples to guarantee the accuracy of the result. An interesting result is depicted in Fig. 8d. In ZIL data set, the approximate algorithms perform better as dimensionality increases. This is because ZIL is correlated and therefore, in spaces with high dimensionality the points are distributed in more cells and thus, the cost of maintaining sample scores and the points in $TOPK$ is reduced. For AHBA and AMSA, this cost dominates the overall cost in a correlated data set and therefore the running time is reduced. Recall, that the number of cells per dimension is fixed, thus in spaces with higher dimensionality, the number of cells increases.

Next, we study the running time versus k. The results are depicted in Fig. 9. As expected, the approximate algorithms are always more efficient than the exact ones. Moreover, AMSA and ADA are always better than AHBA and EVA, respectively. In addition, the performance of AHBA is similar to that of ADA for very small values of k (i.e., $k < 10$). Recall that AHBA reduces the number of exact



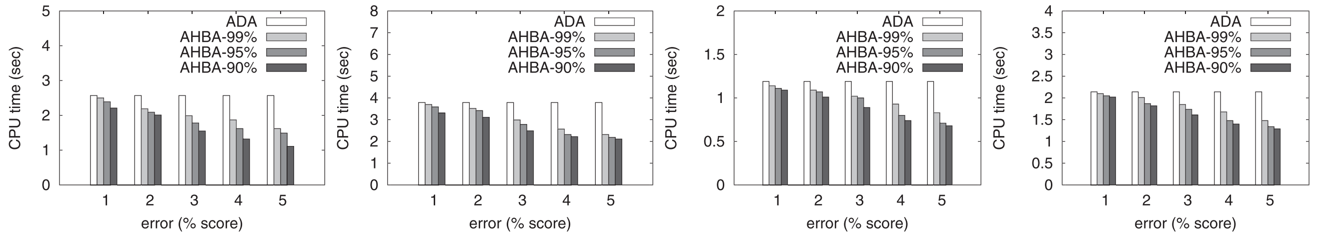
(a) IND (b) ANT (c) FC (d) ZIL

Fig. 8. Running time versus dimensionality (d).



(a) IND (b) ANT (c) FC (d) ZIL

Fig. 9. Running time versus number of answers (k).



(a) IND (b) ANT (c) FC (d) ZIL

Fig. 10. Running time versus error (ε) and probability (δ) of the Hoeffding bound.

score computations. The number of events processed by ADA and AHBA is similar. Therefore, if the number of exact score computations is small, the cost for sample maintenance in this case cannot equalize the cost of exact score computations. In this case, we observe that for very small values of k (<15) the performance of ADA and AHBA is similar. The gap becomes even more significant for larger values of k.

The next experiment studies the behavior of AHBA for different values of the error ε and the probability δ of the Hoeffding bound as defined in (11). Fig. 10 depicts the running time of AHBA and ADA. The parameter ε assumes values from a 1 to 5 percent, whereas δ is set to 0.99, 0.95, and 0.90. As expected, the running time is improved when ε increases and δ decreases, whereas accuracy is not compromised significantly as it is illustrated in Table 2, which shows the accuracy achieved for the ANT and FC data sets, for the same values of ε and δ used in Figs. 10b and 10c, respectively. Accuracy is measured as follows: assume that $TOPK'$ is the result of the approximate algorithm. For each update, the accuracy is computed as $|TOPK \bigcap TOPK'|/k$. Mean accuracy is presented as the average accuracy of all updates, excluding those of the startup phase.

Next, we compare the accuracy of AHBA and AMSA. The results are given in Table 3 for $\varepsilon = 10\%$ and $\delta = 0.9$. As expected, AHBA achieves the best accuracy in all cases in comparison to AMSA. In addition, AHBA may detect a change in $TOPK$ after some updates, in contrast to AMSA which will loose the change completely if the specific point was pruned. Furthermore, AHBA has the nice property that it will miss the points with the lowest scores first. Both AHBA and AMSA have the worst performance for FC data set. In FC, the top-k points are changed frequently; thus, the approximate algorithms achieve poor accuracy since they cannot detect all changes. This phenomenon is minimized as k increases, because the query result is not varied drastically. In general, AMSA offers better response time than AHBA, as shown previously, but shows worse accuracy.

TABLE 2
Mean Accuracy (Percent) for Different Values of ε and δ

	ANT			FC		
ε	δ=0.99	δ=0.95	δ=0.90	δ=0.99	δ=0.95	δ=0.90
1%	100	100	100	99.9	99.8	99.8
2%	100	100	100	99.8	99.8	99.5
3%	100	100	99.9	99.5	99.5	98.9
4%	99.9	99.9	99.9	99.1	98.0	97.8
5%	99.9	99.9	99.9	98.3	97.5	97.4

Next, we measure the number of events that each method processes, the number of exact score computations, and the total number of comparisons (between numbers). Table 4 illustrates the results per 1,000 updates for the ANT data set. It is evident, that ADA reduces significantly the number of the examined events and the exact score computations, especially for large values of k. This happens due to the use of the advanced event computation. Additionally, the number of exact score computations is reduced mainly due to the use of candidate points, as described in Section 5.2. AHBA and ADA process almost the same number of events. The main advantage of AHBA is that it further reduces the number of exact score computations by using sampling. AMSA manages to reduce further both the number of processed events and the number of exact score computations, offering the best running time, but without guarantees.

Next, we examine the storage requirements of the algorithms. EVA uses only the events, whereas ADA, AHBA, and AMSA use events, counters for the advanced event computation and candidates points. Moreover, AHBA maintains samples. The worst memory consumption is

TABLE 3
Mean Accuracy (Percent) versus k ($\varepsilon = 10\%$, $\delta = 0.9$)

	IND		ANT		FC		ZIL	
k	AHBA	AMSA	AHBA	AMSA	AHBA	AMSA	AHBA	AMSA
16	100	100	100	100	81.7	79.6	100.0	99.9
64	100	100	100	99.6	86.9	84.3	99.8	99.6
256	99.9	99.8	99.9	99.2	92.8	91.4	99.6	99.3
512	99.8	99.5	99.8	98.4	94.2	93.8	99.1	98.6

TABLE 4
Cost Analysis per 1,000 Updates (ANT Data Set)

k	algorithm	#events processed	#exact score computations	#comparisons (millions)
16	EVA	1147.3	15.2	26.4
	ADA	1052.7	3.8	12.5
	AHBA	1005.4	0.0	10.6
	AMSA	1000.0	0.0	0.07
64	EVA	201,219.0	22.0	33.7
	ADA	22,247.3	8.7	18.4
	AHBA	22,146.3	0.0	10.8
	AMSA	1000.0	0.0	0.3
256	EVA	201,508.9	50.1	65.3
	ADA	22,361.1	14.7	25.7
	AHBA	22,170.5	0.1	11.6
	AMSA	1000.0	0.0	1.1
512	EVA	802,160.8	296.0	332.2
	ADA	123,942.3	22.6	36.0
	AHBA	123,819.1	0.3	14.0
	AMSA	999.6	0.0	2.8

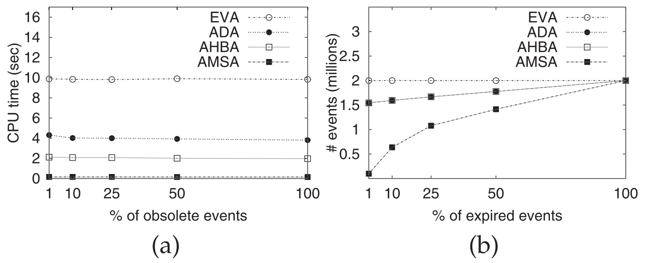


Fig. 11. Running time (a) and number of events (b) versus percentage of maintained obsolete events (ANT).

30.54 MB for AHBA, 30.52 for ADA and AMSA and 22.91 for EVA, for the default set, taking account all data sets. Recall from Section 5.3 that obsolete events have an event processing time larger than the expiration time of the associated point and thus, they may be discarded to reduce memory consumption. However, by discarding obsolete events, the number of points that may be used as reference points is reduced. In all experiments, the algorithms maintain all obsolete events and therefore ADA and AMSA have similar memory consumption. AHBA has a small overhead in memory due to the maintenance of sample points.

It is evident that the major storage cost is due to events. Although the memory requirements are low, we study the effect of obsolete event pruning. In this experiment, a fraction of them is being used while the remaining are pruned. The motivation behind this choice is to check if the elimination of obsolete events has an impact on the running time of the algorithms. The results for the ANT data set are depicted in Fig. 11. The results for IND, FC, and ZIL are similar and thus omitted. Fig. 11a shows the running time with respect to the ratio of obsolete events that each method maintains. For example, ADA has 460,248 obsolete events when 1 percent of them are kept, whereas AMSA has 1,919,567 obsolete events. Since the running time is almost unaffected, it does not pay off to keep a high percentage of obsolete events, because this will impact storage. On the other hand, storage savings are significant, especially for AMSA, as it is depicted in Fig. 11b. Algorithms ADA and AHBA process almost the same number of events, since they both use the advanced event computation technique. AMSA achieves even better storage savings due to the aggressive point elimination technique applied, whereas EVA cannot reduce the number of processed events, because it does not produce obsolete events.

10 Conclusions

Top-k dominating queries have been proposed recently as an alternative to skyline and top-k queries. This query has a number of attractive properties such as: an intuitive ranking function is used, the result is unaffected by dimension scaling and the number of results is bounded by k. This paper is the first study of top-k dominating query processing algorithms in a streaming environment.

The two proposed methods, EVA and ADA consistently outperform the baseline algorithm, whereas ADA shows the best overall performance, being orders of magnitude faster than the brute-force approach. An analysis is performed for the proposed techniques to derive an upper bound for the expected number of exact score computations that may occur. In addition, we have studied two approximate algorithms, AHBA and AMSA, which sacrifice accuracy for faster computation. AHBA is based on sampling and offers probabilistic guarantees regarding the approximation error. On the other hand, AMSA is based on event pruning leading to faster processing with less accuracy compared to AHBA.

The three algorithms, ADA, AHBA, and AMSA, can work in combination. If we have to be strict regarding accuracy then ADA must be used. In case a heavy system load is detected and accuracy may be compromised (but within limits), then AHBA may be activated. Finally, if an even more fast solution is required AMSA may be used. Since all algorithms are based on the event-based framework, they can operate in a synergetic manner, offering significant flexibility.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems," *Proc. ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS)*, pp. 1-16, 2002.

[2] B. Babcock, M. Datar, and R. Motwani, "Sampling from a Moving Window over Streaming Data," *Proc. Thirteenth Ann. ACM-SIAM Symp. Discrete Algorithms (SODA)*, pp. 633-634, 2002.

[3] B. Babcock and C. Olston, "Distributed Top-K Monitoring," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 28-39, 2003.

[4] S. Borzsonyi, D. Kossmann, and K. Stocker, "The Skyline Operator," *Proc. Int'l Conf. Data Eng. (ICDE)*, pp. 421-430, 2001.

[5] C.-Y. Chan, H.V. Jagadish, K.-L. Tan, A.K.H. Tung, and Z. Zhang, "Finding k-Dominant Skylines in High Dimensional Space," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 503-514, 2006.

[6] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with Presorting," *Proc. Int'l Conf. Data Eng. (ICDE)*, pp. 717-719, 2003.

[7] R. Fagin, "Combining Fuzzy Information from Multiple Systems" *J. Computer and System Sciences*, vol. 58, no. 1, pp. 83-99, 1999.

[8] R. Fagin, "Optimal Aggregation Algorithms for Middleware," *Proc. ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS)*, pp. 102-113, 2001.

[9] P. Godfrey, "Skyline Cardinality for Relational Processing: How Many Vectors Are Maximal?," *Proc. Symp. Foundations of Information and Knowledge Systems (FoIKS)*, pp. 78-97, 2004.

[10] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 47-57, 1984.

[11] W. Hoeffding, "Probability Inequalities for Sums of Bounded Random Variables" *J. Am. Statistical Assoc.*, vol. 58, no. 301, pp. 13-30, 1963.

[12] V. Hristidis, N. Koudas, and Y. Papakonstantinou, "PREFER: A System for the Efficient Execution of Multi-Parametric Ranked Queries," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 259-270, 2001.

[13] I.F. Ilyas, G. Beskales, and M.A. Soliman, "A Survey of Top-k Query Processing Techniques in Relational Databases," *ACM Computing Surveys*, vol. 40, no. 4, pp. 1-58, 2008.

[14] W. Kiessling, "Foundations of Preferences in Database Systems," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB)*, pp. 311-322, 2002.

[15] M. Kontaki, A.N. Papadopoulos, and Y. Manolopoulos, "Continuous Top-k Dominating Queries in Subspaces," *Proc. Panhellenic Conf. Informatics (PCI)*, 2008.

[16] H.T. Kung, "On Finding the Maxima of a Set of Vectors" *J. ACM*, vol. 22, no. 4, pp. 469-476, 1975.

[17] X. Lian and L. Chen, "Top-k Dominating Queries in Uncertain Databases" *Proc. 12th Int'l Conf. Extending Database Technology: Advances in Database Technology (EDBT)*, 2009.

[18] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting Stars: The k Most Representative Skyline Operator," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE)*, pp. 86-95, 2007.

[19] Y. Lu, J. Zhao, L. Chen, B. Cui, and D. Yang, "Effective Skyline Cardinality Estimation on Data Streams," *Proc. 19th Int'l Conf. Database and Expert Systems Applications (DEXA)*, pp. 241-254, 2008.
[20] N. Mamoulis, M.L. Yiu, K.H. Cheng, and D.W. Cheng, "Efficient Top-k Aggregation of Ranked Inputs," *ACM Trans. Database Systems*, vol. 32, no. 3, 2007.
[21] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge Univ. Press, 2005.
[22] K. Mouratidis, D. Papadias, S. Bakiras, and Y. Tao, "A Threshold-Based Algorithm for Continuous Monitoring of k Nearest Neighbors," *IEEE Trans. Knowledge and Data Eng.*, vol. 17, no. 11, pp. 1451-1464, Nov. 2005.
[23] K. Mouratidis, S. Bakiras, and D. Papadias, "Continuous Monitoring of Top-k Queries over Sliding Windows," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, pp. 635-646, 2006.
[24] M.J. Osborne and A. Rubenstein, *A Course in Game Theory.* MIT Press, 1994.
[25] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive Skyline Computation in Database Systems" *ACM Trans. Database Systems*, vol. 30, no. 1, pp. 41-82, 2005.
[26] A.N. Papadopoulos, A. Lyritsis, A. Nanopoulos, and Y. Manolopoulos, "Domination Mining and Querying," *Proc. Int'l Conf. Data Warehousing and Knowledge Discovery (DaWaK)*, pp. 145-156, 2007.
[27] D. Skoutas, D. Sacharidis, A. Simitsis, V. Kantere, and T. Sellis, "Top-k Dominant Web Services under Multi-Criteria Matching" *Proc. 12th Int'l Conf. Extending Database Technology: Advances in Database Technology (EDBT)*, 2009.
[28] R.E. Steuer, *Multiple Criteria Optimization: Theory, Computations, and Application.* John Wiley & Sons, 1986.
[29] Y. Tao, L. Ding, X. Lin, and J. Pei, "Distance-Based Representative Skyline," *Proc. IEEE Int'l Conf. Data Eng. (ICDE)*, 2009.
[30] Y. Tao and D. Papadias, "Maintaining Sliding Window Skylines on Data Streams," *IEEE Trans. Knowledge and Data Eng.*, vol. 18, no. 3, pp. 377-391, Mar. 2006.
[31] M. Theobald, G. Weikum, and R. Schenkel, "Top-k Query Evaluation with Probabilistic Guarantees," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, pp. 648-659, 2004.
[32] L. U, K. Mouratidis, and N. Mamoulis, "Continuous Spatial Assignment of Moving Users," *VLDB J.*, vol. 19, pp. 141-160, 2010.
[33] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg, "Reverse Top-k Queries," *Proc. IEEE Int'l Conf. Data Eng. (ICDE)*, 2010.
[34] J.S. Vitter, "Random Sampling with a Reservoir" *ACM Trans. Math. Software*, vol. 11, no. 1, pp. 37-57, 1985.
[35] X. Xiong, M.F. Mokbel, and W.G. Aref, "SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-Temporal Databases," *Proc. IEEE Int'l Conf. Data Eng. (ICDE)*, pp. 643-654, 2005.
[36] M.L. Yiu and N. Mamoulis, "Efficient Processing of Top-k Dominating Queries on Multi-Dimensional Data," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, pp. 483-494, 2007.
[37] M.L. Yiu and N. Mamoulis, "Multidimensional Top-k Dominating Queries," *VLDB J.*, vol. 18, pp. 695-718, 2009.
[38] L. Zhang, Z. Li, M. Yu, and G. Zhao, "New Sampling-Based Summary Structures for Sliding Windows over Data Streams," *Proc. Int'l Conf. Intelligent Computing (ICIC)*, pp. 1242-1249, 2007.

Maria Kontaki received the BS and PhD degrees, both in computer science, from Aristotle University of Thessaloniki, in 2002 and 2009, respectively. Currently, she is a postdoctoral researcher in the Data Engineering Laboratory of the Department of Informatics of Aristotle University, performing research in data streams. Her research interests include stream query processing and mining, and data management issues in sensor networks.

Apostolos N. Papadopoulos received the 5-year Diploma degree in computer science and engineering from the University of Patras and the PhD degree from Aristotle University of Thessaloniki in 1994 and 2000, respectively. Currently, he is an assistant professor in the Department of Informatics at Aristotle University. His research interests include databases, data streams, data mining, and information retrieval. He has served as a track cochair of ACM SAC Database Technologies Techniques and Applications Track (DTTA) 2005 through 2010, as well as a PC member in several International Conferences.

Yannis Manolopoulos received the BEng degree in electrical engineering and the PhD degree in computer engineering, both from the Aristotle University of Thessaloniki in 1981 and 1986, respectively. Currently, he is a professor in the Department of Informatics at the latter university. He has been with the Department of Computer Science of the University of Toronto, the Department of Computer Science of the University of Maryland at College Park and the Department of Computer Science of the University of Cyprus. He has published more than 200 papers in refereed scientific journals and conference proceedings. His work has received over 2,000 citations from more than 450 institutional groups. His research interests include databases, data mining, web and geographical information systems, bibliometrics/webometrics.