Time- and Space-Efficient Sliding Window Top-k Query Processing

KREŠIMIR PRIPUŽIĆ and IVANA PODNAR ŽARKO, University of Zagreb
KARL ABERER, École Polytechnique Fédérale de Lausanne

A sliding window top-k (*top-k/w*) query monitors incoming data stream objects within a sliding window of size w to identify the k highest-ranked objects with respect to a given scoring function over time. Processing of such queries is challenging because, even when an object is not a top-k/w object at the time when it enters the processing system, it might become one in the future. Thus a set of potential top-k/w objects has to be stored in memory while its size should be minimized to efficiently cope with high data streaming rates. Existing approaches typically store top-k/w and candidate sliding window objects in a k-skyband over a two-dimensional score-time space. However, due to continuous changes of the k-skyband, its maintenance is quite costly. Probabilistic k-skyband is a novel data structure storing data stream objects from a sliding window with significant probability to become top-k/w objects in future. Continuous probabilistic k-skyband maintenance offers considerably improved runtime performance compared to k-skyband maintenance, especially for large values of k, at the expense of a small and controllable error rate. We propose two possible probabilistic k-skyband usages: (*i*) When it is used to process all sliding window objects, the resulting top-k/w algorithm is approximate and adequate for processing random-order data streams. (*ii*) When probabilistic k-skyband is used to process only a subset of most recent sliding window objects, it can improve the runtime performance of continuous k-skyband maintenance, resulting in a novel exact top-k/w algorithm. Our experimental evaluation systematically compares different top-k/w processing algorithms and shows that while competing algorithms offer either time efficiency at the expanse of space efficiency or vice-versa, our algorithms based on the probabilistic k-skyband are both time and space efficient.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.3.4 [**Information Storage and Retrieval**]: Systems and Software

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Data stream processing, continuous top-k queries, sliding windows

ACM Reference Format:
Krešimir Pripužić, Ivana Podnar Žarko, and Karl Aberer. 2015. Time- and space-efficient sliding window top-k query processing. ACM Trans. Datab. Syst. 40, 1, Article 1 (March 2015), 44 pages.
DOI: http://dx.doi.org/10.1145/2736701

1. INTRODUCTION

Data stream processing has become an integral part of many applications requiring continuous data processing, such as wireless sensor networks, stock trading, and network monitoring. In contrast to traditional database systems, data stream processing has to cope with high publication rates of data objects, while queries are mostly static and continuous. Therefore, it is imperative to process incoming data streams both time

Authors' addresses: K. Pripužić (corresponding author) and I. Podnar Žarko, Department of Telecommunications, Faculty of Electrical Engineering and Computing, University of Zagreb, Unska 3, HR-10000 Zagreb, Croatia; email: kresimir.pripuzic@fer.hr; K. Aberer, Distributed Information Systems Laboratory, Institute for Core Computing Science, School for Computer and Communication Science, École Polytechnique Fédérale de Lausanne, EPFL-IC-IIF-LSIR, Batiment BC, Station 14, CH-1015 Lausanne, Switzerland.

0362-5915/2015/03-ART1 $15.00
DOI: http://dx.doi.org/10.1145/2736701

efficiently and with a minimal memory footprint, such that the data of interest can be identified and delivered to data destinations, such as end-users or other processes, in near-real time.

We study a particular type of continuous queries which monitor top-k data objects over sliding windows (*top-k / w queries*). The parameter k limits the number of matching data objects within a sliding window of size w to the top-k objects according to a given scoring function. Sliding windows are used to restrict the temporal scope of real-time data processing in the absence of explicit deletions of data objects [Mouratidis and Papadias 2007]. They are commonly defined as either the number of most recent data stream objects (*count-based windows*) or time intervals (*time-based windows*) [Mouratidis et al. 2006]. If a data object is not a top-k object in a query window at the time of its arrival to the system, it can become one later on when higher-ranked objects are dropped from the window while, simultaneously, only lower-ranked objects are arriving. Therefore, the processing engine has to store in memory *candidate objects* with potential to become top-k/w objects in the future. A straightforward approach would maintain all sliding window objects in memory. However, since we are targeting environments with high data publishing rates and queries with potentially large window sizes, the set of top-k/w candidates has to be reduced, especially in scenarios where k is much smaller than the number of sliding window objects.

Top-k/w queries hold a prominent position in the area of data stream processing as they are useful for a number of applications [Jin et al. 2010]. Consider, for example, a large-scale participatory sensing application where users carry their smartphones to measure and report air pollution. A user might be interested to receive daily at most 10 notifications with the highest pollution-level readings over a predefined geographical area, and also at the time when such readings are produced. Another example is monitoring of smart grids where power grid operators need to identify over time a limited number of sites with the largest or lowest energy production. Moreover, top-k/w query processing is required in scenarios involving text retrieval [Haghani et al. 2010; Mouratidis and Pang 2009] of, for example, Twitter messages or Facebook updates in real time, to alleviate the problem of information overload experienced by users.

Although data stream processing has been a particularly active research area in the last years [Chakravarthy and Jiang 2009], there are few solutions for both time- and space-efficient processing of top-k/w queries [Koudas et al. 2004; Mouratidis et al. 2006; Das et al. 2007; Mouratidis and Papadias 2007; Böhm et al. 2007]. Existing top-k/w processing solutions are mainly based on the *dominance property* between data stream objects placed in a two-dimensional *score-time* space. The dominance property states that object a dominates object b iff a has a higher rank[1] than b, and a is younger than b [Mouratidis et al. 2006]. Top-k/w and candidate stream objects, or *nondominated window objects* as we call them in the article, are sliding window objects dominated by at most $k - 1$ other objects. Existing top-k/w processing algorithms typically maintain nondominated window objects over time in a special data structure—*k-skyband* [Papadias et al. 2005]—to produce a resulting top-k data stream. However, due to continuous k-skyband modifications, that is, frequent insertions of incoming objects into the data structure which cause pruning of dominated objects, continuous k-skyband maintenance is extremely costly.

In literature we find two main approaches to efficient k-skyband maintenance: (*i*) *exhaustive indexing* [Mouratidis et al. 2006; Mouratidis and Papadias 2007] of window objects in a regular grid which is tightly coupled with query indexing and (*ii*) *buffering* [Böhm et al. 2007] which stores a FIFO buffer of most recent window objects to improve the k-skyband maintenance procedure. The buffer is used to avoid insertion

[1]Depending on a scoring function, either a smaller or larger score indicates a higher object rank.

of arriving objects with low ranks into k-skybands, since all objects are the youngest and thus nondominated at arrival time. On one hand, exhaustive indexing is efficient in low-dimensional spaces, but not applicable for a wide range of applications that process high-dimensional data objects. Moreover, it has a large memory footprint since all window objects are referenced within a regular grid. On the other hand, the buffering approach requires less memory since it only stores k-skyband and buffer objects in memory, where the buffer size is much smaller than the window size. However, as our experiments show, this approach is relatively slow compared to other approaches, although it does delay the insertion of buffer objects with low ranks into k-skybands.

To overcome the limitations of continuous k-skyband maintenance, we propose an orthogonal approach which is both time and space efficient. It uses *probabilistic k-skyband*, a data structure which is filled by objects that satisfy a probabilistic criterion, to quickly decide from an incoming object score whether to keep the object as a top-k/w candidate or discard it because of low chances to become a future top-k/w object. We propose two possible probabilistic k-skyband usages.

(1) When probabilistic k-skyband is used for processing all sliding window objects, the resulting algorithm is approximate. *Probabilistic candidate pruning algorithm* (PA) is to our knowledge the first approximate algorithm for efficient top-k/w processing of *random-order data streams*. As an approximate algorithm, it may generate both false positive and false negative top-k/w objects with a controllable error rate.
(2) When probabilistic k-skyband processes only buffer objects, it improves the performance of the buffering approach introduced in Böhm et al. [2007]. We call a probabilistic k-skyband containing buffer objects *probabilistic filter* (PF): It is used to quickly filter out buffer objects with low ranks and to prevent their insertion into k-skybands at object arrival time, while such objects may be inserted into k-skybands when exiting the buffer. The resulting algorithm—*strict candidate pruning algorithm with probabilistic filter* (SAPF)—remains exact due to two insertion attempts of buffer objects into k-skybands, as explained further in Section 5.

It is important to distinguish between data stream time and space distributions: a *space distribution* defines how object values are distributed in the attribute space, while a *time distribution* defines object ordering in the stream. A random-order data stream is defined as a data stream for which any permutation of streaming data objects is equally likely to appear in a stream, in other words, its time distribution is random regardless of its space distribution. We can easily generate a random-order data stream from any space distribution by randomly sampling objects in time. Random-order data streams have been identified as a reasonable approximation of real-world data streams (see, for example, Jin et al. [2010]), such as RSS feeds or aggregated data streams stemming from large sensor networks. The random-order data stream model was originally introduced in Munro and Paterson [1978], which is one of the first papers in the field of data stream processing and recently has been used to describe and analyze a number of real-world application scenarios [Guha and McGregor 2006; Chakrabarti et al. 2008a, 2008b; McGregor 2008].

We have observed that lazy periodic pruning of dominated objects from k-skybands has the potential to improve the runtime performance of the *strict candidate pruning algorithm* (SA) which at all times stores strictly nondominated objects in k-skybands. Thus we introduce the notion of a *relaxed k-skyband* which stores nondominated and some dominated objects without affecting top-k/w processing correctness. A relaxed k-skyband is periodically pruned and reduced to a k-skyband. Our exact algorithm based on relaxed k-skybands is called the *relaxed candidate pruning algorithm* (RA). This algorithm can be extended by a buffering approach which uses a probabilistic k-skyband for processing buffer objects, while relaxed k-skybands process all sliding

Table I. Comparison of Top-k/w Algorithms

	Exhaustive indexing [Mouratidis et al. 2006 Mouratidis and Papadias 2007]	Buffering [Böhm et al. 2007]	PA	RAPF
Main characteristics	indexing of all window objects	buffering	approximate pruning of window objects	buffering and approximate pruning of buffered objects
Query-related data structure	k-skyband or top-k set	k-skyband	probabilistic k-skyband	relaxed k-skyband
Filter data structure	N/A	k-skyband	N/A	probabilistic k-skyband
Runtime performance	good	poor	excellent	very good
Memory consumption	poor	very good	good	very good
Accuracy	no errors	no errors	controllable error rate for random-order streams	no errors

window objects. The resulting algorithm—*relaxed candidate pruning algorithm with probabilistic filter* (RAPF)—is the best-performing exact top-k/w algorithm according to our experiments. It can process data objects from all types of stream sources, even highly correlated data streams such as sensor readings from a single sensor. In contrast, PA is best suited for processing random-order data streams. When comparing RAPF to PA, RAPF generates correct results at the expense of a slightly increased but controllable memory consumption and reduced runtime performance compared to PA, while PA may generate false positive or false negative top-k/w objects with a controllable error rate.

Table I contrasts existing approaches with PA and RAPF in terms of their main properties and performance. It lists the data structures used for storing nondominated window objects per each top-k/w query as well as data structures used for processing buffer objects. Further on, it compares the runtime performance, memory footprint, and accuracy of these top-k/w processing algorithms. The table clearly shows that both PA and RAPF overcome performance limitations of existing techniques while in same cases our performance improvements are by orders of magnitude, as demonstrated by the experiments presented in Section 8.

The main article contributions can be summarized as follows.

(1) We introduce a completely novel concept, the probabilistic k-skyband, which stores window objects with high probability to become top-k/w objects. The probabilistic k-skyband exploits a probabilistic criterion which allows to quickly decide whether an incoming object is valuable to be kept in memory. It is used by our approximate top-k/w processing algorithm which runs by orders of magnitudes faster than alternative exact algorithms and with a small and controllable error in terms of false positives and false negatives.
(2) We apply the probabilistic k-skyband to process buffer objects and apply it together with our relaxed k-skyband, resulting in a more efficient exact algorithm (RAPF) for top-k/w processing. In addition, we formally prove that the new algorithm yields correct processing results if the buffer size is at most half the window size.
(3) We analyze results of an experimental evaluation using sliding window k-NN queries on random-order data streams generated from uniform, clustered

Gaussian, and a real sensor dataset. The results show that PA significantly outperforms all exact approaches and offers an improved runtime performance even up to two orders of magnitude for larger values of k with the observed error rate below the theoretical upper bound, while RAPF is superior to existing exact algorithms.

(4) In contrast to existing approaches that focus on specific scoring functions and design algorithms tightly coupled to the applied query indexing structures, we present a formal generic model and implementation for processing top-k/w queries over data streams which is independent of data representation as well as scoring function, and may be applied both with and without query indexing.

The article is organized in the following way. Section 2 presents existing top-k/w processing algorithms and serves as the motivation for our work. In Section 3 we present our formal generic top-k/w stream processing model. Section 4 defines the probabilistic and relaxed k-skybands as well as the algorithms for their computation. The buffering approach and probabilistic k-skyband maintenance of buffer objects which enables filtering of (relaxed) k-skyband objects are introduced in Section 5. Section 6 introduces additional data structures needed to efficiently process multiple top-k/w queries. Both time and space complexity of the introduced algorithms is presented in Section 7, while in Section 8 we experimentally evaluate the performance of our algorithms and compare them to those defined in Mouratidis and Papadias [2007] and Böhm et al. [2007]. Section 9 briefly reviews related work and compares existing solutions to our approach. Section 10 concludes the article and identifies directions for future work.

2. BACKGROUND

The first paper addressing the problem of exact top-k/w processing over append-only data streams presents two algorithms supporting monotone scoring functions, namely the *top-k monitoring algorithm* (TMA) and *skyband monitoring algorithm* (SMA) [Mouratidis et al. 2006]. To reduce the set of candidate objects maintained in memory, the authors introduce a vital property for top-k/w processing—*dominance* in a two-dimensional score-time space. This property enables pruning of non-candidate objects because an object dominated by k or more than k objects from a query window cannot become a top-k/w object in the future. In their subsequent paper, Mouratidis and Papadias [2007] present two algorithms conceptually similar to TMA and SMA, *conceptual partitioning monitoring over sliding windows* (CPM/w) and *skyband nearest neighbor* (SNN), which are developed specifically for top-k/w queries supporting distance scoring functions.

The previously listed algorithms divide the task of continuous top-k/w processing into the following subtasks: (1) continuous maintenance of per-query data structures storing top-k and candidate data objects, and (2) indexing of queries in a regular grid. In particular, TMA and CPM/w continuously maintain only top-k objects within a query structure, while SMA and SNN maintain a set of all top-k/w and candidate objects in a k-skyband. In situations when a query data structure contains less than k objects, these algorithms initiate the computation of top-k objects from scratch. Thus they require exhaustive indexing of data objects within a regular grid. Additionally, a regular grid is used for query indexing to identify the *subspace of interest* for each query and to efficiently recompute top-k objects from scratch. The subspace of interest is bounded by a threshold which is defined as the score of an object with rank k for a given query. It is used to make a quick decision on whether an arriving object needs to be processed for a query or not because a query monitors exclusively those objects which fall within its subspace of interest.

Consider an example of a top-1 *nearest-neighbor* (NN) query q over a count-based window of size 6 which is defined as a point in a two-dimensional Euclidean space, while objects $o_1, o_2, \ldots, o_{11}$ represent points from the same space ordered by their time

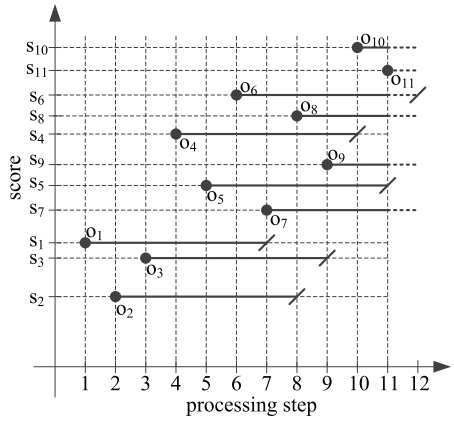


Fig. 1. Example objects in the two-dimensional score-time space.

Table II. CPM/w and SNN Processing Steps for the Example in Figure 1

processing step	CPM/w		SNN	
	top-k	threshold	k-skyband	threshold
1	o_1	s_1	o_1	s_1
2	o_2	s_1	o_2	s_1
3	o_2	s_1	o_2,o_3	s_1
4	o_2	s_1	o_2,o_3	s_1
5	o_2	s_1	o_2,o_3	s_1
6	o_2	s_1	o_2, o_3	s_1
7	o_2	s_1	o_2,o_3	s_1
8	o_3	s_3	o_3	s_1
9	o_7	s_7	o_7	s_7
10	o_7	s_7	o_7	s_7
11	o_7	s_7	o_7	s_7

of appearance. Figure 1 shows these objects in a two-dimensional score-time space. At each processing step i, object o_i enters the query window and expels object o_{i-6} from the window. Table II shows the content of query data structures and query threshold values for CPM/w and SNN when processing the example objects.

At processing step 1, object o_1 appears and is automatically added to the top-k data structure of CPM/w query q. This data structure contains only top-k objects from the query window. Additionally, the query threshold is set to s_1, that is, to the score of object o_1. This threshold is used for query indexing in the grid due to the fact that all objects outside of the sphere of radius s_1 cannot become top-1 objects while o_1 is in the query window. Therefore, query indexing reduces the number of appearing objects that a top-k/w query needs to process by neglecting those objects which are certainly not of interest for query q. Figure 2 demonstrates how query q is indexed during this processing step. All cells which are either encompassed or intersected by the indexing threshold s_1 (cells in dark and light gray) need to be monitored. Each object that appears in these cells will be inserted into the top-k data structure associated with q only if its score is smaller than the query threshold. For example, when object o_2 appears at processing step 2, it replaces object o_1 in the top-k data structure because o_2 dominates o_1 since it is both younger and has a smaller score than o_1. Although the

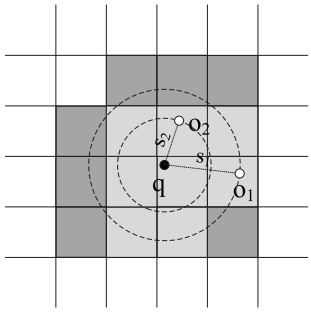


Fig. 2. CPM/w and SNN query indexing.

query sphere has obviously been reduced, CPM/w does not re-index the query to reduce the number of costly grid operations. In Figure 2 we see that the sphere of radius s_2 is completely covered by light-gray cells and thus the query could be re-indexed. Such a lazy approach to query indexing enables runtime performance improvement because it reduces costly grid operations at the expanse of processing objects which are potentially not top-k. Such approach shows good results in practice if score calculation is simple because CPM/w only needs to compare the score of an arriving object with the score of the k-th object in the top-k data structure. Therefore, in our example, subsequent objects $o_3, o_4, \ldots, o_7$ are ignored as they either fall out of the q's subspace of interest or their scores are larger than s_2. At processing step 8, object o_2 is expelled from the query window by object o_8 and thus removed from the top-k data structure, which becomes empty. Now the set of top-k objects has to be recomputed from scratch: o_3 becomes the top-1 object while the query indexing threshold changes to s_3. The query is not re-indexed since $s_3 < s_1$. When o_9 appears, it expels o_3 from the query window and the set of top-k objects has to be recomputed from scratch once again: Object o_7 becomes the top-1 object and its score s_7 becomes the new query threshold. Now the query q has to be re-indexed because $s_7 > s_1$. Finally, subsequent objects s_{10} and s_{11} are ignored by the query since their scores are larger than the current query threshold.

In contrast to CPM/w, SNN tries to reduce the number of recomputations of top-k objects from scratch by keeping a k-skyband of query window objects in memory, instead of only top-k objects. Let us investigate how SNN processes objects from the previous example given in Figure 2. When object o_1 appears at processing step 1, it is added to the empty k-skyband associated with query q, while the query indexing threshold is set to s_1. When object o_2 appears, it is added to the k-skyband structure from which it prunes the dominated object o_1. The query indexing threshold changes and is set to s_2 but, analogous to CPM/w, the query is not re-indexed. When object o_3 appears, it is within the monitored hypersphere and thus added to the query k-skyband. On the contrary, subsequent objects $o_4, o_5, \ldots, o_7$ are ignored by the query as they either fall out of the q's subspace of interest or their scores are larger than s_2. At processing step 8, object o_2 is removed from the k-skyband because it is no longer within the query window, and object o_3 becomes the top-1 object. Again, the query is not re-indexed since $s_3 < s_1$. When o_9 appears, it expels o_3 from the query window and the k-skyband has to be recalculated from scratch. Thereafter, o_7 becomes the top-1 object and the query expands. Due to the expansion, the query is re-indexed with the query threshold value set to s_7.

Böhm et al. [2007] have noticed that an arriving object is always nondominated by other window objects because it is the youngest irrespective of its rank, and thus all arriving objects have to be inserted into the k-skyband unless a query indexing mechanism is in place. This is quite inefficient because objects with low ranks soon become dominated by objects with higher ranks. Thus Böhm et al. [2007] *filter* less

Table III. SASF Processing Steps for the Example in Figure 1

processing step	SASF			
	query k-skyband	strict filter	threshold	FIFO buffer
1	o_1	o_1	s_1	o_1
2	o_2	o_2	s_1	o_2,o_1
3	o_2	o_2,o_3	s_1	o_3,o_2,o_1
4	o_2	o_2,o_3	s_1	o_4,o_3,o_2
5	o_2	o_3,o_5	s_1	o_5,o_4,o_3
6	o_2,o_3	o_5	s_5	o_6,o_5,o_4
7	o_2,o_3,o_7	o_7	s_5	o_7,o_6,o_5
8	o_3,o_7	o_7	s_5	o_8,o_7,o_6
9	o_7	o_7,o_9	s_5	o_9,o_8,o_7
10	o_7	o_9	s_9	o_{10},o_9,o_8
11	o_7	o_9	s_9	o_{11},o_{10},o_9

relevant and recent data objects by avoiding their insertion into a query k-skyband at the time of appearance. In addition to a query k-skyband storing nondominated window objects, they associate an additional data structure to each query, termed the *approximate* k-skyband. Each approximate k-skyband contains only most recent objects from a special FIFO buffer that is shared by all queries. The size of a FIFO buffer is expected to be much smaller than the query window size. An object is inserted into a query k-skyband either upon appearance (i.e., when entering the FIFO buffer) if it is among top-k objects in the approximate k-skyband or when exiting the buffer if it is not dominated by the objects currently residing in the approximate k-skyband.

To use a consistent algorithm naming scheme throughout the article, further on we refer to this algorithm as the *strict candidate pruning algorithm with a strict filter* (SASF), where *strict filter* denotes the approximate k-skyband, as it is basically a k-skyband of strictly nondominated buffer objects. Note that the buffer and associated strict filters enable a different query indexing strategy compared to CPM/w and SNN: The query indexing threshold is defined as the score of the k-th object from the strict filter.

Let us investigate the buffering approach by Böhm et al. [2007] through the same example. Table III shows the content of data structures and query threshold values for SASF when processing the example objects in Figure 1. In this example we define a FIFO buffer of size 3 such that the buffer contains only 3 most recent objects at each processing step. When object o_1 appears at processing step 1 it is automatically added to the query k-skyband, strict filter, and buffer, while s_1 becomes the new query threshold. The query indexing strategy of SASF is different from CPM/w and SNN, since SASF does not ignore objects appearing in the indexed cells whose score is larger than the query threshold. Such objects are used to fill the strict filter. When object o_2 appears at processing step 2, it is added to the query k-skyband, query filter, and buffer, and expels the dominated object o_1 from both the query k-skyband and query filter. Analogous to CPM/w and SNN, the query threshold update and related query re-indexing can be postponed to reduce the number of costly grid operations. In the next processing step, object o_3 appears and is added into to the buffer and query filter. The former insertion is automatic, while the latter is related to the fact that o_3 is within the monitored cells. At processing step 4, object o_4 is inserted into the buffer and expels o_1 from it. Since o_4 is not within the monitored cells, it is not added to the filter. When object o_5 appears, it is automatically added to the buffer and, although s_5 is larger than the current indexing threshold s_1, it is added to the filter because it is within the monitored cells. Figure 3 shows this situation in the two-dimensional attribute space. We see that o_5 is outside

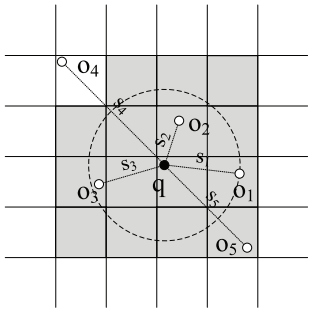


Fig. 3. Another example of query indexing.

of the query sphere of interest, but within the monitored cells. Such objects are ignored by CPM/w and SNN, but not by SASF.[2] Additionally, object o_2 is expelled both from the buffer and filter. The same happens with o_3 when o_6 appears. Furthermore, at this processing step o_5 becomes the only object in the filter, the query threshold is set to s_5, and the query has to be re-indexed since $s_1 < s_5$. Most importantly, o_3 is inserted to the query k-skyband because its score is smaller than the new query threshold and it was previously not inserted into the k-skyband. When object o_7 appears, it is added to the buffer and also to both the query filter and k-skyband due to the fact that its score is smaller than s_5. Actually, it dominates and thus expels o_5 from the filter. During processing step 8, object o_2 is expelled from the query window and has to be removed from the query k-skyband. When this happens, o_3 becomes the top-1 object. Similarly, o_7 replaces o_3 as the top-1 object when object o_9 arrives. At step 9, o_9 is added to the filter as it is within the monitored cells (not shown in figures) and the query has to be re-indexed in step 10 when o_7 is expelled from the buffer and related filter.

To conclude, both CPM/w and SNN require exhaustive indexing of data objects for the purpose of recomputing top-k objects from scratch when there are less than k objects within the top-k list. SNN tries to reduce the number of recomputations by storing potential top-k candidates in a query k-skyband. Both CPM/w and SNN are tightly coupled with query indexing to improve the runtime performance. The buffering approach does not require exhaustive indexing of data objects, but rather uses two k-skybands, the first for all window objects and the second for buffer objects, to disable objects with low ranks to enter the first k-skyband. The buffering approach is formally defined in Section 5.

3. TOP-K/W PROCESSING MODEL

In this section we present a generic data stream model for top-k/w processing and introduce the problem of maintaining a minimal set of data objects needed to produce a correct answer to a top-k/w query. The model is built assuming the data stream processor architecture depicted in Figure 4, which is in accordance with the generic architecture proposed in Golab and Özsu [2003]. A single certain and append-only data stream represents the processor input: each incoming data object is immutable after entering the system and associated with a timestamp denoting its time of arrival. Additionally, an object is considered valid for a top-k/w query while it belongs to the query sliding window. Without loss of generality, the model assumes count-based sliding windows and may be extended in a straightforward manner to support time-based sliding windows. The processor accepts new top-k/w queries and query updates, and outputs result streams, where each result stream is associated with an active top-k/w

[2]This is the main reason why query filter is named *approximate k-skyband* in Böhm et al. [2007].

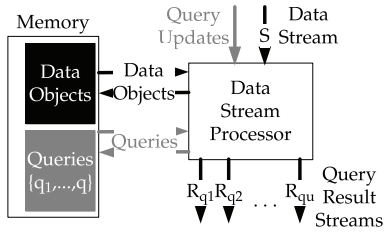


Fig. 4. Data stream processor architecture.

Table IV. Notation

Symbol	Description
S	a data stream
o_i	an object appearing at step i (i-th object in S)
q	a top-k/w query
W_i^q	objects in a window of q at step i
T_i^q	top-k objects from the window of q at step i
R_i^q	result stream associated with q at step i
$O \overset{s}{\triangleright} o$	objects from a set O that according to a scoring function s are ranked higher than o
$O \overset{s}{\blacktriangleright} o$	objects from O that according to a scoring function s dominate o
kS_i^q	k-skyband of q at step i
pS_i^q	probabilistic k-skyband of q at step i
rS_i^q	relaxed k-skyband of q at step i

query. We assume queries reference future data objects, that is, those objects entering the system after query activation. As each incoming data object is seen only when entering the system, unless explicitly stored in memory, the processor maintains only a subset of objects from the input stream needed for efficient top-k/w processing.

Table IV provides the notation used throughout the article.

3.1. Model Definition

Given a data stream object $o_i = \{c_i, t_i\}$ with *sequence number* i represented by its content c_i and time of appearance t_i, we define a data stream as an infinite set of objects ordered by their time of appearance.

Definition 3.1 (*Data Stream*). A data stream $S = \{o_1, o_2, \ldots, o_i, \ldots\}$ is an infinite set of data stream objects ordered by increasing times of their appearance such that $\forall o_i, o_j \in S : i < j \Leftrightarrow t_i < t_j$.

In other words, data stream objects are unique in their time of appearance, implying that a single object may enter the system at a point in time. Besides being ordered by their time of appearance, data stream objects may be ranked based on their content, and thus we define a scoring function for calculating object-specific scores.

Definition 3.2 (*Scoring Function*). Let S be an incoming data stream. We define the scoring function $s : S \mapsto \mathbb{R}$ as any function which assigns a score for all $o_i \in S$.

Scoring functions are application specific and their explicit definition depends completely on the application scenario in which the processor is used. We are particularly interested in those distance, aggregation, and relevance scoring functions which are

commonly used in the application domains that we envision for our processor. Note that the model supports generic scoring functions because we do not impose any specific constraints on the scoring function definition, such as monotonicity, which is frequently assumed by many top-k processing techniques [Ilyas et al. 2008]. Our assumption is that scoring functions are time independent, that is, $s(o_i) = s(c_i)$ and that objects from S have unique scores with respect to a query, implying that they can be ranked consistently.[3] Scoring functions may assign ranks to objects either in descending or ascending order of their scores. Without loss of generality, we assume the applied scoring function (e.g., k-NN) is such that smaller scores are preferable to larger.

Hereafter we define top-k/w queries supported by our stream processing model.

Definition 3.3 (*Top-k/w Query*). We define a continuous top-k query over a count-based sliding window of size n as a quintuple $q = \{s, k, n, t_A, t_C\}$, where s is a scoring function, $k, n \in \mathbb{N}$, while t_A and t_C are two points in time such that $t_A < t_C$.

A top-k/w query is defined by a scoring function s associated with two additional parameters: the query window size n (count-based window), and parameter k denoting the number of top objects from the window that can be inserted in the query result stream. Furthermore, as top-k/w queries are continuous, they have a predefined *time of activation* t_A and *time of cancellation* t_C. We say that the query is active within the period $(t_A, t_C]$.

A data object is within the query window if the object enters the processor after query activation and is among the n most recent data objects. This is formally stated by the following definition. As the model supports count-based sliding windows, a query window can be seen as a sequence of snapshots between two consecutive object appearances. Thus *step* i is the time period $[t_i, t_{i+1})$ between appearances of objects o_i and o_{i+1}.

Definition 3.4 (*Objects in a Query Window*). We define the set of data objects in the window of an active query q at step i as follows:

$$W_i^q \stackrel{\text{def}}{=} \{o_j : o_j \in S \wedge i \geq j > i - n \wedge t_C \geq t_j > t_A\}. \tag{1}$$

Only objects within a query window are of interest for top-k/w processing, and thus our model supports ad-hoc queries referencing future objects. An important point in the object's lifetime is $t_i^D = t_{i+n}$, a point in time when o_i is dropped from the window because a new object o_{i+n} has just appeared and expelled o_i from the window.

Definition 3.5 (*Top-k/w Objects*). We define the set of top-k data objects in the query window of an active query q at step i as follows:

$$T_i^q \stackrel{\text{def}}{=} \{o : o \in W_i^q \wedge |W_i^q \stackrel{s}{\rhd} o| < k\}, \tag{2}$$

where $W_i^q \stackrel{s}{\rhd} o \stackrel{\text{def}}{=} \{o' : o' \in W_i^q \wedge s(o') < s(o)\}$ is the set of objects from W_i^q that are ranked higher than o according to s.

In other words, a data object o is a top-k/w object for a query q if o is within the current window of q and there are less than k higher-ranked objects than o in this window. Another important point in the top-k/w object's lifetime is $t_i^R = t_j$, a point in time when o_i becomes a top-k/w object for the first time, that is, $o_i \in T_j^q$.

Finally, we define the query result stream which the processor produces for each active top-k/w query.

[3]A tie-breaking criterion can be used to give preference to more recent objects since the time of appearance is unique for each object.

Definition 3.6 (*Result Stream*). We define a result stream for a query q at step i as the set of q's top-k/w objects $R_i^q \stackrel{\text{def}}{=} \{o_j : o_j \in T_l^q \wedge l \leq i\}$ ordered by their increasing times t_j^R.

3.2. Problem Statement

Let us now discuss when an object o_i is inserted into a query result stream, that is, what is the point in time t_i^R. According to Definition 3.5, an object within the window of a top-k/w query becomes its top-k/w object when there are less than k higher-ranked objects within the window. This can happen either: (1) at t_i when o_i enters the processor and there are less than k higher-ranked objects within the query window, or (2) at a later point in time t_j when o_j appears but is not a top-k/w object, while a top-k/w object $o_{j-n} \in T_{j-1}^q$ is dropped from the query window and o_i has the highest rank among top-k/w candidate objects.

The implementation of the first scenario is quite straightforward: An incoming data object is compared against the list of current top-k/w objects for a query and, in case its rank is higher than the rank of the k-th object, it is added to the result stream. This requires continuous maintenance of top-k/w objects in memory because, although these objects have already been inserted into the result stream, they are continuously compared to arriving objects.

The second scenario requires a more elaborate solution, as has already been recognized [Böhm et al. 2007; Mouratidis and Papadias 2007; Jin et al. 2010]: The processor needs to instantly fill in the slot of a dropped top-k/w object because the newly arrived object is not among the current top-k/w objects. The empty slot is filled in with an object that currently has the highest rank among the top-k/w candidates. A simple solution would be to keep all data objects within the query window in memory and regard them as candidate data objects. However, this solution would have a large memory footprint since the query window size can in general be much larger than k.

The problem we address in this article is the following: Given a deterministic and unbounded data stream, we wish to continuously detect top-k/w data stream objects such that we maintain a sufficient set of candidate objects in memory which allows efficient processing of data stream objects. In other words, we are interested in top-k/w algorithms that are primarily time efficient, but at the same time do not sacrifice the space efficiency.

As stated in Section 1, current state-of-the-art algorithms use the *dominance property* to identify data objects which are irrelevant for a top-k/w query such that they cannot affect its result stream. Hereafter we state it formally.

THEOREM 3.7. *A data object $o_j \in W_i^q$ cannot become a top-k/w object of an active query q at steps $l = i+1, i+2, \ldots, j+n-1$ if there are k or more dominators of o_j in W_i^q. More formally,*

$$\forall o_j \in W_i^q : |W_i^q \overset{s}{\blacktriangleright} o_j| \geq k \Rightarrow \nexists l > i : o_j \in T_l^q, \tag{3}$$

where $W_i^q \overset{s}{\blacktriangleright} o_j \stackrel{def}{=} \{o_m : o_m \in W_i^q \overset{s}{\triangleright} o_j \wedge t_m > t_j\}$ is a set of objects from W_i^q that dominate o_j for q at step i.

PROOF. Assume to the contrary that $\exists l : j+n > l > i \wedge o_j \in T_l^q$ and that $|W_i^q \overset{s}{\blacktriangleright} o_j| \geq k$. Since all objects in $W_i^q \overset{s}{\blacktriangleright} o_j$ are more recent than o_j, we have $|W_i^q \overset{s}{\blacktriangleright} o_j| \geq k \Rightarrow \forall l : j+n > l > i \wedge |W_l^q \overset{s}{\triangleright} o_j| \geq k$, which is, according to expression (2), in contradiction with our assumption. □

A data object o_j is not a candidate data object for q if, at any step while o_j is within the window, it is *dominated* by k or more than k objects from the query window, that is, there are at least k younger objects with higher rank than o_j. In the rest of the article we say that such an object is *dominated* for query q, and otherwise when o_j is dominated by less than k objects, we say it is *nondominated* for query q.

Using Theorem 3.7, we define query *k-skyband* that contains all nondominated and only nondominated candidate objects for a query.

Definition 3.8 (*k-Skyband*). We define k-skyband $kS_i^q \subseteq W_i^q$ associated with an active query q at step i as the set of all data objects from the query window which are dominated by less than k data objects. More formally,

$$kS_i^q \stackrel{\text{def}}{=} \{o : o \in W_i^q \wedge |W_i^q \overset{s}{\blacktriangleright} o| < k\}. \tag{4}$$

The k-skyband has to be maintained in memory over time to produce exact answers to a top-k/w query, and thus represents the basic building block for top-k/w processing algorithms.

4. TOP-K/W PROCESSING ALGORITHMS

This section presents two novel top-k/w processing algorithms. The first algorithm, named the probabilistic candidate pruning algorithm (PA), relies on a probabilistic criterion to decide whether an object, at the time of its appearance, has potential to become a top-k/w object for a query. PA maintains a probabilistic k-skyband for each query containing objects whose probability of becoming a top-k/w object is above a predefined threshold, and thus introduces a controllable error rate. The second algorithm, which stores top-k/w and candidate objects in a relaxed k-skyband, is the relaxed candidate pruning algorithm (RA). It is an exact algorithm based on lazy pruning of dominated objects from a relaxed k-skyband. The relaxed k-skyband stores all nondominated objects for a query but, contrary to a k-skyband, the relaxed k-skyband may also store some dominated objects. To simplify the discussion, we present PA and RA when processing a single top-k/w query $q = \{s, k, n, t_A, t_C\}$. Section 6 deals with processing of multiple top-k/w queries.

4.1. Probabilistic Candidate Pruning Algorithm

In practice, most of the objects from a query k-skyband will never become top-k/w objects since the k-skyband size is in general much larger than k. For each incoming object, PA computes the probability that an object may become a top-k/w object. If the probability is above a predefined threshold, the object is maintained in the *probabilistic k-skyband*, or is discarded otherwise. PA is designed to process *random-order data streams*, while it generates a large error rate in the case of data streams with highly time-correlated objects, as we explain in the end of this section. In comparison to exact top-k/w processing algorithms, the main drawback of PA is that, as an approximate algorithm, it may introduce both false positive and false negative top-k/w objects.

In this section we first present the mathematical background which allows us to calculate the object's probability to become a top-k object in the future, and subsequently define the PA.

4.1.1. Mathematical Background. To explain the mathematical background of PA, let us assume the stream processor has unbounded memory which, at every processing step, can store all objects from the current query window. We define the *score list* L_i of q at step i as a list of data objects within the window of q at i which are ordered by their descending ranks $L_i \stackrel{\text{def}}{=} W_i^q$.

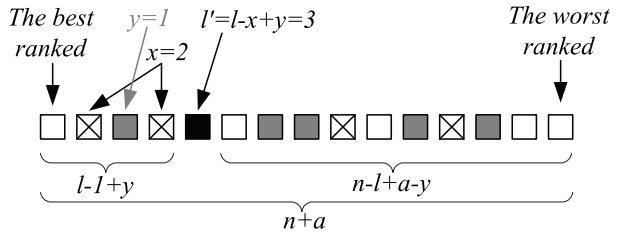


Fig. 5. An example score list after several object appearances.

Let us assume that at step i a new object o_i arrives such that its initial rank is l in the score list. The rank of this object will change during time as new objects arrive while the old ones are dropped from the window. Our goal is to find the probability that o_i will become a top-k/w object of q. The initial rank l of o_i may have the following properties:

(1) top-k rank: o_i is a top-k/w object at step i,
(2) good rank: o_i is not a top-k/w object at i but has probability $p \geq \sigma$ to become a top-k/w object later on while still in the window, and
(3) poor rank: o_i is not a top-k/w object at i and has low probability $p < \sigma$ to become a top-k/w object later on.

If l is a top-k or good rank, o_i is stored in memory and referenced in the probabilistic k-skyband. Otherwise, if l is a poor rank, o_i is ignored. To simplify the discussion, we assume the following: (1) we know only the scores (i.e., ranks) of objects in L, and not their actual sequence numbers in the data stream and (2) the cardinality of L is n at all points in time, that is, we will ignore the initial period after query activation when L is not full. Let a new rank of o_i in L_j be $l' = l - x + y$ at a later step $j > i$, where x is the number of objects from L_j with a higher rank than o_i at step i that have been dropped from the window during steps between i and j, and y is the number of objects that have arrived in the meantime and have a higher rank than o_i.

Figure 5 shows an example score list at step j, $j > i$. Object o_i is shown as a black square, objects that have arrived before o_i are shown as empty or checked squares, and objects that have arrived after o_i are shown as gray squares. Note that objects represented by checked squares have previously expired and thus are not elements of L_j. However, we have to take these objects into account because our assumption is that any permutation of streaming data objects is equally likely in the stream. The initial rank of o_i is $l = 4$, and Figure 5 shows the score list at step j. The new rank l' of o_i at step j is 3 since two objects with higher ranks than o_i have been dropped while only one object ranked higher than o_i has arrived.

LEMMA 4.1. *For data objects from a random-order stream, the probability that x objects have higher ranks than o_i out of d objects which have been dropped during steps $i+1, i+2, \ldots, i+d$ is*

$$p_{drop}(l, n, d, x) = \frac{\binom{l-1}{x}\binom{n-l}{d-x}}{\binom{n-1}{d}}, \tag{5}$$

where $l \leq n$ and $x \leq d$.

PROOF. We have to find the probability that, among d dropped objects, x are ranked higher than o_i. Since every object in L_i, except for o_i, has equal probability to be dropped, there are $\binom{n-1}{d}$ different possibilities to choose d dropped objects from $n-1$ potentially dropped objects. Additionally, x of d dropped objects are ranked higher than o_i, and thus there are $\binom{l-1}{x}$ different possibilities to choose x dropped objects from

$l-1$ potentially dropped objects that are ranked higher than o_i, and $\binom{n-l}{d-x}$ different possibilities to choose $d-x$ objects from $n-l$ potentially dropped objects with lower ranks than o_i. □

LEMMA 4.2. *For data objects from a random-order stream, the probability that y objects have higher ranks than o_i out of a objects which have arrived during steps $i+1, i+2, \ldots, i+a$ is*

$$p_{arrival}(l,n,a,y) = \frac{n}{n+a} \cdot \frac{\binom{l-1+y}{y}\binom{n-l+a-y}{a-y}}{\binom{n+a-1}{a}}, \tag{6}$$

where $l \leq n$ and $y \leq a$.

PROOF. We have to find the probability that, among a arrived objects, y are ranked higher than o_i. This is a conditional probability $p(E_1|E_2) = p(E_1 \cap E_2)/p(E_2)$, where event E_1 is "y of a arrived objects are ranked higher than o_i", and event E_2 is "the rank of o_i in L_i is l". The probability of event E_2 is $1/n$ because there are n possible ranks of o_i in L_i. The $p(E_1 \cap E_2)$ is the probability that the rank of o_i in L_i is l and that among a arrived objects y are ranked higher than o_i. There are $\binom{n+a-1}{a}$ different possibilities to choose a arrived objects from $n+a-1$ equally possible ranks in L_j. Additionally, we know that the rank of o_i in L_j is $l' = l-x+y$, where x is the number of dropped among higher-ranked objects. Therefore, there are $\binom{l-1+y}{y}$ different possibilities to choose y arrived objects from $l-1+y$ objects that are ranked higher than o_i. Analogously, there are $\binom{n-l+a-y}{a-y}$ different possibilities to choose $a-y$ arrived objects from $n-l+a-y$ objects that are ranked lower than o_i, and $n+a$ possibilities to choose the rank of o_i in L_j. Thereby, the probability $p(E_1 \cap E_2)$ is given by $(\binom{l-1+y}{y}\binom{n-l+a-y}{a-y})/(\binom{n+a-1}{a}(n+a))$, and then we have $p(E_1|E_2) = p(E_1 \cap E_2)/p(E_2) = n \cdot (\binom{l-1+y}{y}\binom{n-l+a-y}{a-y})/(\binom{n+a-1}{a}(n+a))$. □

The following lemma defines the theoretical criterion which we can use to decide whether an incoming object needs to be maintained in memory.

LEMMA 4.3. *Let q be a top-k/w query over a count-based window of size n, and let o_i be a data object with rank $l > k$ within the window of q at step i. For data objects from a random-order data stream, an upper bound on the probability $p_{topk}(l,n,k)$ that o_i will become a top-k/w object before it is dropped from the window of q is*

$$p_{theory}(l,n,k) = \sum_{s=1}^{n-1} \sum_{l'=1}^{k} \sum_{y=0}^{l'-1} [p_{drop}(l,n,s,l-l'+y) \cdot p_{arrival}(l,n,s,y)], \tag{7}$$

where $l \leq n$.

PROOF. The probability that object o_i has rank l' during step j is $p_{rank}(l,l',n,s) = \sum_{y=0}^{l'-1}[p_{drop}(l,n,s,l-l'+y) \cdot p_{arrival}(l,n,s,y)]$, where $s = j-i$, since we need to sum all products of probabilities $p_{drop}(l,n,s,x) \cdot p_{arrival}(l,n,s,y)$ for which $l' = l-x+y$. Furthermore, we get the probability that object o_i is a top-k during step j by summing probabilities of top-k ranks $l' = 1,2,\ldots,k$: $p_{topkstep}(l,k,n,s) = \sum_{l'=1}^{k} p_{rank}(l,l',n,s)$. For each processing step $j = i+1, i+2, \ldots, i+n-1$, let E_j be the event "o_i is a top-k/w object at step j". Let $p_{topk}(l,n,k) = p(\sum_{j=i+1}^{i+n-1} E_j)$ be the probability of o_i being a top-k/w object at any step. Obviously, events $E_{i+1}, E_{i+2}, \ldots, E_{i+n-1}$ are dependent and thus we can use inequality $p(\sum_{j=i+1}^{i+n-1} E_j) \leq \sum_{j=i+1}^{i+n-1} p(E_j)$ to bound the value of $p_{topk}(l,n,k)$. Finally, since $p(E_j) = p_{topkstep}(l,k,n,s)$ we get $p_{topk}(l,n,k) \leq \sum_{s=1}^{n-1}\sum_{l'=1}^{k}\sum_{y=0}^{l'-1}[p_{drop}(l,n,s,l-l'+y) \cdot p_{arrival}(l,n,s,y)] = p_{theory}(l,n,k)$. □

Since significant computational power is required to calculate $p_{theory}(l, n, k)$ in practice, hereafter we present its upper bound $p_{practice}(l, n, k)$ that we use for this purpose.

Note that during the last processing step, that is, when $j = i + n - 1$, just before o_i is dropped from the window, all other objects that have arrived before o_i are already dropped and thus its rank in L_j exclusively depends on the ranks of objects which have arrived after it. Therefore, using expression (6), we can calculate the probability that object o_i has rank l' at step $j = i + n - 1$ as

$$p_{last}(l', l, n) = p_{arrival}(l, n, n-1, l'-1) = \frac{n}{2n-1} \cdot \frac{\binom{l+l'-2}{l'-1}\binom{2n-l-l'}{n-l'}}{\binom{2n-2}{n-1}} = \frac{n}{2n-1} \cdot \frac{\binom{n-1}{l'-1}\binom{n-1}{l-1}}{\binom{2n-2}{l+l'}}. \quad (8)$$

The following theorem defines the basis of our probabilistic criterion which we use to decide whether an incoming object needs to be maintained in memory.

THEOREM 4.4. *Let q be a top-k/w query over a count-based window of size n, and let o_i be a data object within the window of q at step i and with rank $l > L$.*[4] *For data objects from a random-order data stream, an upper bound on the probability $p_{theory}(l, n, k)$ that o_i will become a top-k/w object before it is dropped from the window of q is*

$$p_{practice}(l, n, k) = \frac{n^2}{4n-2} \cdot \sum_{l'=1}^{k} \frac{\binom{n-1}{l'-1}\binom{n-1}{l-1}}{\binom{2n-2}{l+l'}}, \quad (9)$$

where $l \leq n$ and $l' \leq n$.

PROOF. Let E_j be the event "o_i is a top-k/w object at step j", and $s = j - i$. Lemmas A.1 and A.2 in the article appendix prove that the probability $p(E_j)$ as a discrete function of step j is monotonically increasing and convex for all initial ranks $l > L$. Since o_i is not a top-k/w object upon its arrival, for such ls we have $p(E_i) = 0$. Then, using the triangle shown in Figure 6, we bound the value of each $p(E_j)$ with its projection $\frac{j-i}{n-1} \cdot p(E_{i+n-1})$ on the triangle diagonal: $\sum_{j=i+1}^{i+n-1} p(E_j) \leq \frac{n}{2} \cdot p(E_{i+n-1})$. Furthermore, $p(E_{i+n-1})$ is equal to $\sum_{l'=1}^{k} p_{last}(l', l, n)$, where $p_{last}(l', l, n)$ is defined by Eq. (8). Finally, we have $p_{practice}(l, n, k) = \frac{n}{2} \cdot p(E_{i+n-1}) = \frac{n^2}{4n-2} \cdot \sum_{l'=1}^{k} \frac{\binom{n-1}{l'-1} \cdot \binom{n-1}{l-1}}{\binom{2n-2}{l+l'}}$, and $p_{practice}(l, n, k) \geq \sum_{j=i+1}^{i+n-1} p(E_j) = p_{theory}(l, n, k)$. □

As stated in the following definition, a probabilistic k-skyband stores only query window objects with (good) initial ranks such that $l \leq L$ or such that the sum of probabilities $p_{practice}(l', n, k)$ for $l' = l_j + 1, l_j + 2, \ldots, n$ is larger than the predefined probability of error σ.

Definition 4.5 (*Probabilistic k-Skyband*). We define probabilistic k-skyband $pS_i^q \subseteq W_i^q$ associated with an active query q at step i as

$$pS_i^q \stackrel{\text{def}}{=} \left\{ o_j : o_j \in W_i^q \wedge \left[\sum_{l'=l_j+1}^{n} p_{practice}(l', n, k) \geq \sigma \vee l \leq L^{\S} \right] \right\}, \quad (10)$$

where l_j is the rank of o_j in W_j^q at step j.

[4]Where $L = \frac{3 \cdot n - 4 \cdot k + 2 \cdot k \cdot n + 3 + \sqrt{3 \cdot (-8 \cdot k^2 \cdot n + 4 \cdot k^2 + 8 \cdot k \cdot n^2 + 4 \cdot k \cdot n - 4 \cdot k - 5 \cdot n^2 - 2 \cdot n + 3)}}{2 \cdot n + 2}$ as defined in the article appendix.

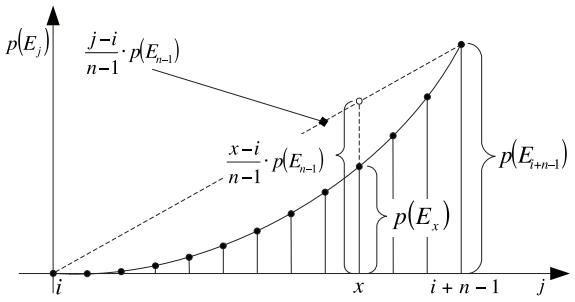


Fig. 6. Bounding $p(E_j)$, the probability that o_i is a top-k/w object at step j.

Table V. Maximal Number of Object Candidates in a Probabilistic k-Skyband (*limit*) for $\sigma = 0.001$

$n \setminus k$	1	2	5	10	20	50	100	200	500
10^3	18	21	26	32	40	56	72	91	106
10^4	22	25	30	37	46	65	86	116	172
10^5	25	28	34	41	51	72	95	128	192
10^6	28	32	38	46	56	78	103	138	207

For data stream processing scenarios with specific σ, we can compute the number of objects to be maintained in the probabilistic k-skyband using Eq. (9). Hereafter, we sketch the algorithm we use for this purpose.

To calculate this, we use a specific property of the values of $p_{practice}(l, n, k)$: after a certain rank l_h, we have $p_{practice}(l'' - 1, n, k) > 2 \cdot p_{practice}(l'', n, k)$ for the probabilities of its successive ranks (i.e., for $\forall l'' : l_h < l'' \leq n$). Therefore we have that $\forall l'' : l_h \leq l'' < n : \sum_{l'=l''+1}^{n} p_{practice}(l', n, k) < 2 \cdot p_{practice}(l'', n, k)$ since this geometric series converges. We iteratively process ranks $l = L + 1, L + 2, \ldots$ and calculate.[5] $p_{practice}(l, n, k)$ using Eq. (9). We stop at the first rank $l = l_c$ for which $p_{practice}(l_c, n, k) < \sigma/2$. If $l_c \geq l_h$ we have $p_{practice}(l_c, n, k) + \sum_{l''=l_c+1}^{n} p_{practice}(l'', n, k) < 2 \cdot p_{practice}(l_c, n, k) < \sigma$, and therefore the maximal number of candidates in a probabilistic k-skyband is equal to $limit = l_c - 1$. Otherwise (i.e., when $l_c < l_h$) we iterate through ranks $l_h - 1, l_h - 2, \ldots, l_c$ until we find a first rank l_w for which it holds that $\sum_{l''=l_h-1}^{l_w} p_{practice}(l'', n, k) + 2 \cdot p_{practice}(l_h, n, k) > \sigma$, and then $limit = l_w$. The number of objects in a probabilistic k-skyband is equal to $k + limit$.

Table V shows candidate limits of probabilistic k-skybands for typical values of parameters k and window sizes n when $\sigma = 0.001$. We can see that the number of candidates is much smaller than window size n, and decreases significantly compared to n for large values of n. For small values of parameter k, the number of candidates is larger in the probabilistic k-skyband than in the k-skyband since the expected size of the latter is $k[ln(\frac{n}{k}) - 1]$ for random-order data streams, as shown in Zhang [2008].

4.1.2. Algorithm Description. The probabilistic k-skyband is implemented by two self-balancing binary trees, a *top-k tree* and *candidate tree*, storing objects sorted by descending ranks. Note that complex data structures (R-tree and quadtree) have shown an inadequate processing performance since they cannot cope with frequent object

[5]Note that summands in formula (9) may have large numerators and denominators. To avoid a potential arithmetic overflow error, we sort numerators and denominators by their decreasing values in two lists, and then calculate the quotient gradually by polling numerators and denominators of similar values from the list to keep the intermediate result around the value of 1, and then divide it with the rest of the denominators.

insertions and deletions, and therefore we have decided to use binary trees for object maintenance. The *top-k tree* contains the k highest-ranked data objects, while the *candidate tree* stores objects with lower ranks for which criterion $\sum_{l'=l_j+1}^{n} p_{approx}(l', n, k) < \sigma$ from Definition 4.5 holds. Each object is associated with its score and sequence number. Additionally, we use a *time tree* which sorts objects by ascending sequence numbers to efficiently detect dropped objects that need to be removed from the k-skyband.

Three additional attributes are associated with each query: (1) the maximal number of candidates in a probabilistic k-skyband (*limit*), (2) *threshold*, and (3) a number of *additional* candidate objects. The limit is numerically determined (e.g., values in Table V) upon query activation using Eq. (9) from Theorem 4.4 and the criterion from Definition 4.5, as previously specified, while the threshold is used for quick detection of candidates with good ranks. The threshold value is initially set to the largest possible score value since the probabilistic k-skyband is empty, while, when the probabilistic k-skyband is full, it is set to the score of the candidate tree tail object. Additional candidate objects are used together with query indexing to improve its performance. The three attributes are used in the subsequent algorithm and are its only connection with the mathematical background from Section 4.1.1.

ALGORITHM 1: PA: Process an incoming object

```
1: //step 1
2: if s(o) > threshold then
3:    abort because o has a poor rank
4: else
5:    //step 2
6:    if s(o) < s(toptree.tail) then
7:       //step 3
8:       report o as a top-k/w object;
9:       add o to toptree;
10:      if toptree.size ≥ k then
11:         move toptree.tail from toptree to candtree;
12:      end if
13:   else
14:      //step 3
15:      add o to q.candtree;
16:   end if
17: end if
18: add o to timetree;
19: //step 4
20: if candtree.size ≥ limit + additional then
21:    if candtree.size > limit + additional then
22:       remove candtree.tail from candtree and timetree;
23:    end if
24:    set threshold = candtree.tail.score; //new tail
25: end if
```

As shown in Algorithm 1, the processing of an incoming data object is done in four steps. We first compare the object score with the threshold (line 2) and abort the procedure (line 3) when it is higher than the threshold because the object rank is poor, or continue with the second step otherwise (lines 5–15). In the second step we compare the object score with the score of the top-k tree tail (line 6). In the third step, the object is inserted either into the top-k tree or into the candidate tree. If the object score is smaller than the score of the top-k tree tail, it is reported as a top-k/w object and added to the top-k tree (lines 8 and 9). Additionally, if the size of the top-k tree becomes larger

than k, we move the top-k tree tail to the candidate tree (line 11). If the object's initial rank is good, it is added to the candidate tree (line 14). PA adds both top-k tree and candidate tree objects to the time tree (line 18) because the candidate tree can contain dominated objects which can be dropped from the query window. However, this does not create a significant overhead as the candidate tree size is comparable to k, as shown in Table V. When the candidate tree size reaches the predefined limit (line 20) in the fourth step, a new threshold value is set (line 24) and, if it is overgrown (line 21), the tail object is removed from the candidate tree and time tree (line 22).

ALGORITHM 2: PA: Remove a dropped top-k/w object

```
1: if s(o) < s(toptree.tail) then
2:   remove o from toptree and timetree;
3:   if candtree is not empty then
4:     h ← candtree.head;
5:     move h from candtree to toptree;
6:     if h has not been previously reported then
7:       report h as a top-k/w object;
8:     end if
9:   end if
10: else
11:   remove o from candtree and timetree;
12: end if
13: if candtree.size < limit then
14:   reset threshold;
15: else
16:   increase threshold;
17: end if
```

PA checks the oldest object in the time tree upon each new object arrival to find out whether this object is dropped from the query window. When this happens, the dropped object is removed from the probabilistic k-skyband. The algorithm for removing dropped objects is presented in Algorithm 2. Since we add only probabilistic k-skyband objects to the time tree, a dropped object is either a top-k object or a candidate tree object (line 1). In the latter case we remove it from both the candidate and time trees (line 11). In the former case (lines 2–9), we remove it from the top-k and time trees (line 2), and move the candidate tree head to the top-k tree to fill in the empty slot in the top-k tree (line 5). If the moved object has previously not been inserted into the query result stream (line 6), it is reported as a top-k/w object (line 7). After each object removal, we check the number of objects left in the candidate three (line 13). If there are less than $limit$ candidates, we have to reset the threshold value (line 14) to the largest possible score. Otherwise, we increase the threshold value (line 16) to avoid its costly reset. This value is increased by an absolute score difference divided by 2 of those two objects from the candidate tree with the lowest ranks.

Finally, let us return to the example of a top-1/6 query introduced in Section 2 to show the processing steps of PA so that it is comparable to CPM/w, SNN, and SASF. Table VI shows the content of data structures and query threshold values for PA when processing the example objects from Figure 1. For this example, we assume $additional = 1$ and $limit = 3$, where the latter is calculated using the values of parameter $k = 1$, query window size $n = 6$, and a selected probability of error σ. When object o_1 appears at processing step 1, it is automatically added to the top-k tree. The indexing threshold is undefined because the candidate tree is still not full. When object o_2 appears at processing step 2, it is added to the top-k tree while o_1 is pushed to the candidate tree.

Table VI. PA Processing Steps for the Example in Figure 1

processing step	PA		
	probabilistic k-skyband		
	top-k tree	candidate tree	threshold
1	o_1		∞
2	o_2	o_1	∞
3	o_2	o_3, o_1	∞
4	o_2	o_3, o_1, o_4	∞
5	o_2	o_3, o_1, o_5, o_4	s_4
6	o_2	o_3, o_1, o_5, o_4	s_4
7	o_2	o_3, o_7, o_5, o_4	s_4
8	o_3	o_7, o_5, o_4, o_8	s_8
9	o_7	o_5, o_9, o_4, o_8	s_8
10	o_7	o_5, o_9, o_8	$s_8 + \frac{1}{2} \cdot (s_8 - s_9)$
11	o_7	o_9, o_8, o_{11}	∞

In processing steps 3 and 4, objects o_3 and o_4 appear and are added into to the candidate tree. When object o_5 appears, it is added to the candidate tree and the query is indexed for the first time with indexing threshold s_4. Next, object o_6 is ignored as it either falls out of the q's subspace of interest or its score is smaller than s_4. At processing step 7, object o_1 is removed from the candidate tree because it is no longer within the query window and the query is reindexed with the following threshold value $s_4 + \frac{1}{2} \cdot (s_4 - s_5)$. Then object o_7 is added to the candidate tree and the threshold value is set to s_4, while query reindexing can be postponed to reduce the number of costly grid operations. In the next processing step, o_2 is dropped from the query window and has to be removed from the top-k tree. An empty space in the top-k tree is filled with object o_3 and the threshold value increases to $s_4 + \frac{1}{2} \cdot (s_4 - s_5)$. Next, object o_8 is added to the candidate tree and the query threshold is set to s_8, while reindexing is again postponed. At step 9, object o_3 is dropped from the window and top-k tree, and thus o_7 becomes a top-k object. After this, o_9 is added to the candidate tree. At processing step 10, object o_4 is dropped from the window and the query is reindexed with the threshold value set to $s_8 + \frac{1}{2} \cdot (s_8 - s_9)$. Next, object s_{10} is ignored by the query since its score is larger than the current query threshold. Finally, at processing step 11, object o_5 is dropped from the query window, the threshold value is reset, and object o_{11} is added to the candidate tree.

4.1.3. Analysis of PA Error Rate. In the next lemma we bound the expected error rate of PA in the case of random-order data streams.

LEMMA 4.6. *The expected error rate of PA is bounded by a predefined probability of error σ such that, per N processed objects streaming for a random-order stream, it generates less than $\sigma \cdot \frac{N}{n}$ false negative and less than $\frac{3}{2} \cdot \sigma \cdot \frac{N}{n}$ false positive objects for a top-k/w query with window size n.*

PROOF. An incoming object has the probability $\frac{k+limit}{n}$ to be added to the probabilistic k-skyband, and the probability $1 - \frac{k+limit}{n} = \frac{n-(k+limit)}{n}$ to be ignored at arrival time. Therefore, the probability of an event A defined as "an incoming object is ignored by the probabilistic k-skyband" is equal to $p(A) = \frac{n-(k+limit)}{n}$. The probability of an ignored object to become a top-k/w object is expected to be less than $\frac{\sigma}{n-(k+limit)}$ since in (10) we define σ as the limit of the sum of these probabilities for all ignored ranks. Therefore, the probability of a conditional event $B|A$ defined as "an incoming object which is ignored by the probabilistic k-skyband becomes a top-k/w object" is $p(B|A) \leq \frac{\sigma}{n-(k+limit)}$.

Then for the probability of event $A \cap B$ defined as "an incoming object is first ignored by the probabilistic k-skyband and then it becomes a top-k/w object" we have $p(A \cap B) = p(A) \cdot p(B|A) \leq \frac{n-(k+limit)}{n} \cdot \frac{\sigma}{n-(k+limit)} = \frac{\sigma}{n}$. Finally, per N processed objects, it is expected that we have $N \cdot p(A \cap B) \leq \sigma \cdot \frac{N}{n}$ missed top-k/w objects (i.e., false negative objects). When an object is missed by the probabilistic k-skyband, a lower-ranked (and younger) object becomes a top-k/w object (i.e., false positive). Additionally, each new object that appears with rank k also becomes a false positive. The expected number of such objects is $\frac{n-(limit+1)}{2} \cdot \frac{1}{n} = \frac{1}{2} \cdot (1 - \frac{limit+1}{n}) \approx \frac{1}{2}$ since it is expected that at most $\frac{n-(limit+1)}{2}$ objects still have to be processed before the false negative is dropped from the window, and each of them has probability $\frac{1}{n}$ to appear with rank k. Note that a false negative object is older than consequential false positive objects, and thus will not generate any additional false positives. □

Hereafter, we analyze why PA generates errors in the case of streams with time-correlated distributions. A stream of temperature readings from a single sensor can be used as an example since temperature typically slowly rises to its peak in the afternoon, and then slowly falls to its lowest value during the night. Of course, there can be also several local minima during the day when a thick cloud appears between the sensor and sun. Let us assume that a query defines such a scoring function which equals the current temperature value while it prefers larger scores. First, let us analyze the case when temperature readings are monotonically increasing in time. Every new reading is ranked higher than all previous readings in the query window. This is the best-case scenario for top-k/w processing algorithms since new objects are the highest ranked when entering the processor and thus quickly expel older objects from the query k-skyband (its size is thus small). In the case of SA and RA, this happens because new objects quickly over-dominate the old ones. In the case of PA, new objects are added to the probabilistic k-skyband while the old ones are expelled due to the fact that this type of k-skyband has a limited size (equal to the previously defined *limit* value). PA does not generate errors since a situation when an expelled object can become top-k/w is not possible. Now, let us analyze the case when temperature readings are monotonically decreasing in time. This is the worst-case scenario for the top-k/w processing algorithms since all appearing objects have to be kept in the k-skyband because they will become top-k/w objects just before being dropped from the window, when all older and higher-ranked objects are already dropped. In this case, the size of SA and RA k-skybands is equal to the query window size. Note that in both monotonically increasing and decreasing scenarios, all appearing objects become top-k/w objects at some point in time, either immediately after appearance in the monotonically increasing scenario or eventually in the monotonically decreasing scenario. Thus the resulting stream equals the incoming stream, and the top-k/w processor is not able to filter such streams. In the worst-case scenario, PA will miss almost all top-k/w objects because it has dropped them from the probabilistic k-skyband since their initial ranks calculated when entering the processor were estimated as poor.

A question arises on whether it is possible to calculate the PA error rate for data streams with correlated time distribution. The main idea behind PA is that it is possible to calculate the probability that an object o_i, which appears at step i and has rank $l > k$ within the window of query q, will eventually become a top-k/w object for q. This probability can be calculated for random-order streams as shown in Section 4.1.1, but unfortunately cannot be calculated for data streams with an arbitrary correlated time distribution.

PA guarantees a bounded error rate even if the stream space distribution is time varying, as long as its time distribution stays random. PA has problems processing data

streams that are correlated in time, or when the data stream time distribution changes from random to correlated. To avoid this problem, we can employ a hybrid PA/RAPF.[6] algorithm which monitors a small buffer of recent stream objects and switches the processing algorithm from approximate PA to exact RAPF when such time distribution change is detected, and vice-versa. Such a hybrid algorithm can guarantee a bounded error rate while offering the best processing efficiency. However, the implementation of a hybrid algorithm is a nontrivial task since it requires special techniques for data stream monitoring which should be adapted for each use-case and application scenario. For this reason, it is out of the scope of this article, but represents a possible direction for future work on adaptive top-k/w processing.

4.2. Relaxed Candidate Pruning Algorithm

RA maintains a relaxed k-skyband and shows better processing performance compared to continuous maintenance of a k-skyband as applied in Böhm et al. [2007]. This comes at the expense of a slightly increased but controllable memory consumption. RA uses lazy pruning of dominated objects when the k-skyband size reaches a predefined limit. The pruning process reduces a relaxed k-skyband to a k-skyband.

Definition 4.7 (*Relaxed k-Skyband*). We define relaxed k-skyband rS_i^q associated with an active query q at step i as a set of objects from the query window W_i^q which is a superset of the corresponding k-skyband: $kS_i^q \subseteq rS_i^q \subseteq W_i^q$.

Analogously to PA, the relaxed k-skyband is implemented by two self-balancing binary trees, a *top-k tree* and *candidate tree*, storing objects sorted by descending ranks. The top-k tree contains the k highest-ranked data objects, while the candidate tree stores objects with lower ranks. Each object is associated with its score and sequence number. Additionally, we use a *time tree* which sorts objects by ascending sequence numbers to efficiently detect dropped objects that need to be removed from the relaxed k-skyband. Only top-k/w objects are added to the time tree because, as shown in the following lemma, other objects are already dominated when being dropped from the query window.

LEMMA 4.8. *A data object o_j which is in the k-skyband of an active query q with window size n at step $j+n-1$ is also among q's top-k/w objects. More formally,*

$$\forall o_j \in kS_{j+n-1}^q \Rightarrow o_j \in T_{j+n-1}^q. \tag{11}$$

PROOF. All data objects within the window of q at step $j+n-1$ are more recent than $o_j \in kS_{j+n-1}^q$ and thus will be dropped later from the window. Therefore, from expression (4) we have $|W_{j+n-1}^q \overset{s}{\blacktriangleright} o_j| = |W_{j+n-1}^q \overset{s}{\triangleright} o| < k$, and then from expression (2) it directly follows that $o_j \in T_{j+n-1}^q$. □

Two additional attributes are associated with an RA top-k/w query: (1) *pruning coefficient* γ, and (2) *candidate tree limit*. The pruning coefficient represents the percentage of the acceptable overhead of the candidate tree size compared to a k-skyband, and is used to calculate the candidate tree limit expressed as the number of objects in the candidate tree. When this limit is reached, RA triggers the pruning of dominated objects in the relaxed k-skyband. After each pruning, we set the candidate tree limit to $(1+\gamma)$ multiplied by the current candidate tree size. The initial value of the candidate tree size is set to be a few times larger than k. The following lemma defines the size of such a bounded relaxed k-skyband.

[6]Note that RAPF is our exact algorithm which we define in the following sections.

LEMMA 4.9. *The expected number of objects in the relaxed k-skyband of an RA query is equal to* $(1+\frac{\gamma}{2})\cdot n' - \frac{\gamma}{2}\cdot k$ *in the case of a random-order stream, where* $n' = k\cdot ln(\frac{n}{k})$ *is the expected number of objects in the corresponding k-skyband.*

PROOF. The pruning of a relaxed k-skyband is triggered when the number of candidates reaches its candidate tree limit. After each pruning, the relaxed k-skyband has the same number of objects as the corresponding k-skyband since all dominated objects are pruned from it. If the expected number of objects in the k-skyband is n', the number of objects in the relaxed k-skyband varies between n' and $k+(1+\gamma)\cdot(n'-k)$. The former size corresponds to the moment immediately after a pruning, while the latter size corresponds to the moment just before the next pruning. Therefore the average number of objects in a relaxed k-skyband is $\frac{n'+k+(1+\gamma)\cdot(n'-k)}{2} = \frac{2\cdot n'+\gamma\cdot(n'-k)}{2} = (1+\frac{\gamma}{2})\cdot n' - \frac{\gamma}{2}\cdot k$. Finally, in Zhang [2008] it is shown that in the case of random-order streams $n' = k\cdot ln(\frac{n}{k})$. □

The processing of an incoming data object is done in three steps as shown in Algorithm 3. We first compare the object score to the score of the top-k tree tail object (line 2). If the object score is smaller than the score of the top-k tree tail, the object is reported as a top-k/w object (line 4) and added to the top-k and time tree (line 5), while the top-k tree tail is moved to the candidate tree and removed from the time tree (lines 7–9). Otherwise, the object is added to the candidate tree (line 12). When the number of objects in the candidate tree becomes larger than the predefined candidate tree limit (line 15), the pruning process is triggered (line 16) and the new candidate tree limit is calculated (line 17). After each pruning, only nondominated data objects, that is, k-skyband objects, remain in the relaxed k-skyband.

ALGORITHM 3: RA: Process an incoming object

```
1: //step 1
2: if s(o) < s(toptree.tail) then
3:    //step 2
4:    report o as a top-k/w object;
5:    add o to toptree and timetree;
6:    if toptree.size ≥ k then
7:       t ← toptree.tail;
8:       move t from toptree to candtree;
9:       remove t from timetree;
10:   end if
11: else
12:   add o to candtree; //step 2
13: end if
14: //step 3
15: if candtree.size ≥ limit then
16:   prune dominated objects from candtree;
17:   set limit = (1 + γ) · candtree.size;
18: end if
```

To efficiently prune dominated objects from the candidate tree, we use an auxiliary priority queue called the *dominator tree* which, during pruning, stores k most recent objects sorted by their sequence numbers. The pruning process is performed as in the DominatingSet algorithm proposed in Tsaparas et al. [2003] to prune dominated tuples while processing top-k join queries. As shown in Algorithm 4, after the initial filling of the dominator tree with top-k tree objects (lines 2 and 3), RA traverses the candidate tree from head to tail so that each encountered object is either inserted into the dominator tree, in case this object is younger than the dominator tree head (line 8),

ALGORITHM 4: RA: Prune dominated objects

```
1: domtree ← {∅}
2: for all o in toptree do
3:    add o to domtree;
4: end for
5: for all o in candtree do
6:    h ← domtree.head;
7:    if t_o > t_h then
8:       add o to domtree;
9:       remove h from domtree;
10:   else
11:      remove o from candtree;
12:   end if
13: end for
```

or removed from the candidate tree (line 11) since it is dominated by k objects in the dominator tree.

ALGORITHM 5: RA: Remove a dropped top-k/w object

```
1: remove o from toptree;
2: repeat
3:    h ← candtree.head;
4:    if h is not null then
5:       remove h from candtree;
6:    end if
7: until h is null or h is in the window;
8: if h is not null then
9:    add h to toptree and timetree;
10:   if h is previously not reported then
11:      report h as a top-k/w object;
12:   end if
13: end if
```

When removing a dropped object, upon each object arrival, RA checks whether the oldest object in the time tree, that is, a top-k object, is dropped from the query window. The object needs to be removed from the relaxed k-skyband, while a candidate tree object head takes its position in the top-k tree. As shown in Algorithm 5, the object is first removed from the top-k tree (line 1) and then we identify the head object from the candidate tree which takes its place (lines 2–7): We first check that the candidate tree is nonempty, and then ensure the current head object is within the window since the candidate tree may contain objects that have already been dropped from the window. These are removed from the candidate tree, either while searching for a head object within a query window (line 5) or during the subsequent pruning, as defined in Algorithm 4. If the candidate tree is nonempty (line 8), the identified head object is moved to the top-k tree and time tree to fill in the empty slot (line 9). Additionally, if the moved object has previously not been inserted into the query result stream (line 10), it is instantly reported as a top-k/w object (line 11).

Finally, let us again return to the example of a top-1/6 query introduced in Section 2 to show the processing steps of RA so that it is comparable to CPM/w, SNN, SASF, and PA. Table VII shows the content of query data structures when processing the example objects from Figure 1. We do not show query threshold values since RA cannot

Table VII. RA Processing Steps for the Example in Figure 1

processing step	RA	
	relaxed k-skyband	
	top-k tree	candidate tree
1	o_1	
2	o_2	o_1
3	o_2	o_3, o_1
4	o_2	o_3, o_1, o_4
5	o_2	$o_3, \cancel{o_1}, o_5, \cancel{o_4}$
6	o_2	o_3, o_5, o_6
7	o_2	o_3, o_7, o_5, o_6
8	o_3	o_7, o_5, o_8, o_6
9	o_7	o_5, o_9, o_8, o_6
10	o_7	$\cancel{o_5}, o_9, \cancel{o_8}, \cancel{o_6}, o_{10}$
11	o_7	o_9, o_{11}, o_{10}

be indexed without additional modifications which we explain in the following sections. For this example, we assume the initial candidate tree limit is equal to $limit = 4 * k = 4$ and that the pruning coefficient is $\gamma = 0.5$. When object o_1 appears at processing step 1, it is automatically added to the top-k tree. After this, when object o_2 appears at processing step 2, it is added to the top-k tree while o_1 is pushed to the candidate tree. In the processing steps 3 and 4, objects o_3 and o_4 are added to the candidate tree. When object o_5 appears, it is added to the candidate tree, which becomes oversized and thus the pruning process is started. It prunes dominated objects o_1 and o_4 from the candidate tree. After this, the value of the limit becomes $limit = (1 + \gamma) \cdot 2 = 3$, where 2 is the current size of the candidate tree. Another pruning of the candidate tree is initiated after o_6 enters it, but this process does not prune any objects, while the limit changes to the value $limit = \lceil (1 + \gamma) \cdot 3 \rceil = 5$. In the processing step 7, object o_7 is also added to the candidate tree. Afterwards, o_2 is dropped from the query window and has to be removed from the top-k tree. An empty space in the top-k tree is filled with object o_3, and object o_8 is added to the candidate tree. At step 9, object o_3 is dropped from both the query window and top-k tree, and thus o_7 becomes a top-k object, while o_9 is added to the candidate tree. The adding of object o_{10} to the candidate tree starts another pruning process which prunes objects o_5, o_8, and o_6. The limit also changes and becomes $limit = (1 + \gamma) \cdot 2 = 3$. Finally, object o_{11} is added to the candidate tree.

5. OBJECT BUFFERING AND FILTERING

In practice, as already stated in Section 2, arriving objects with low ranks are non-dominated, but soon become dominated by higher-ranked and more recent objects. Thus, Böhm et al. [2007] avoid the insertion of objects with low ranks into k-skybands by employing a special FIFO buffer in combination with an approximate k-skyband, an additional data structure associated with each top-k/w query. The approximate k-skyband is a k-skyband containing only buffer objects and is used to filter out arriving objects with low ranks so that they are not inserted into a query k-skyband, but rather reside within the approximate k-skyband while within the buffer. Thus we call the approximate k-skyband a *strict query filter*.

We extend the contribution of Böhm et al. [2007] by proving that any subset of buffer objects can be used as a query filter. Thus we propose that our probabilistic k-skyband containing buffer objects is used as a query filter. The resulting *probabilistic query filter* has the potential to significantly improve runtime performance of the buffering

approach introduced in Böhm et al. [2007]. Hereafter, we formally define the extended version of the buffering approach to top-k/w processing.

A data stream S is associated with a FIFO buffer which stores b most recent data objects similarly to Definition 3.4, where n is replaced by b: $B_i \stackrel{\text{def}}{=} \{o_j : o_j \in S \wedge i \geq j > i - b\}$. Typically $b \ll n$, and the buffer is represented in memory as a singly-linked list of objects added by ascending sequence numbers, and is shared by all queries. We associate an auxiliary data structure to each query, called a *query filter*. The query filter is associated with the query's scoring function and parameter k but, in contrast to a k-skyband or a relaxed k-skyband associated with the query, it refers only to b objects from the buffer instead of n objects from the query window.

The query filter is used as an intermediary between the buffer and query k-skyband. It maintains top-k and candidate objects from the buffer, sorted by descending ranks, and makes two attempts to insert an object into a query k-skyband. The first attempt occurs upon object arrival while the second attempt happens just before the object is dropped from the buffer. In the first attempt, an object is inserted into a query k-skyband only if it is a top-k object in the query filter. In the second insertion attempt, if the object is not among top-k objects in a query filter, it is surely dominated for the query and thus may safely be ignored. The logic behind the two insertion attempts is formally stated in the following theorem which shows that an object cannot become a top-k object while in the buffer iff it was not a top-k object when entering the buffer and if buffer size is less than half of the window size. This theorem is originally included in Böhm et al. [2007] and we include it here for completeness.

THEOREM 5.1. *Let B be a recent buffer of size b and let o_i be a data object which arrives to the system at step i when a query q is active. If o_i is not among top-k objects in a set $O \subset B_i \overset{s}{\triangleright} o_i$, and $b \leq \frac{n+1}{2}$, o_i cannot become a q's top-k/w object while in the buffer (i.e., at steps $l = i, i+1, \ldots, i+b-1$).*

PROOF. If every element from $O \subset B_i \overset{s}{\triangleright} o_i$ is still in the window of q at step $i+b-1$, it holds that $\forall l : i+b > l \geq i \Rightarrow |O \subset B_i \overset{s}{\triangleright} o_i \subseteq W_i^q \overset{s}{\triangleright} o_i| \geq k$, and then from expression (2) we have $\nexists l : i+b > l \geq i \wedge o_i \in T_l^q$. Let o_{i-b+1} be the oldest object in B_i which will be expelled from B at step $i+1$. If o_{i-b+1} is still in the window of q at step $i+b-1$, all other objects in B_i are also within q's window since they are more recent than o_{i-b+1}. Finally, using expression (1) we get $i-b+1+n-1 \geq i+b-1$ and thus $b \leq \frac{n+1}{2}$. □

Therefore if $b \leq \frac{n+1}{2}$, objects delayed in the buffer cannot become top-k/w objects while in the buffer. In practice, buffer size is typically chosen such that it is much smaller than n and larger than k.

From Theorem 3.7 we conclude that an object is dominated for the query when it is expelled from the buffer iff its rank is lower than the rank of any subset of k objects left in the buffer. Thus any subset of buffer objects can be used as a query filter.

The algorithm defined in Böhm et al. [2007] uses two k-skybands associated with a query, one k-skyband containing top-k and candidate objects from the query windows and the other k-skyband with buffer objects for object filtering. Thus we call this algorithm the *strict candidate pruning algorithm with strict filter* (SASF). To improve the performance of the query filter, we introduce the *probabilistic filter* (PF) which maintains a probabilistic k-skyband of buffer objects while reusing the probabilistic criterion defined in Eq. (7) and Definition 4.5 by replacing the window size n with buffer size b. The probabilistic k-skyband can be used as a query filter with both SA- and RA-based queries. The algorithms behind the former and latter queries are called *strict candidate pruning algorithm with probabilistic filter* (SAPF) and *relaxed candidate*

pruning algorithm with probabilistic filter (RAPF), respectively. The SA-based query from Böhm et al. [2007] uses the k-skyband as a query data structure, while our RA-based query uses the relaxed k-skyband. The following lemma proves that RAPF offers a correct result stream, that is, it is an exact algorithm.

LEMMA 5.2. *Relaxed candidate pruning algorithm with probabilistic filter (RAPF) is an exact top-k/w processing algorithm when the buffer size is* $b \leq \frac{n+1}{2}$.

PROOF. To prove this lemma we have to show that any relaxed k-skyband rS_j^q associated with a probabilistic filter pS_j^b always (i.e., $\forall j$) contains all top-k/w objects from the RAPF query window of size n. Assume to the contrary that at a step $j : i \leq j < i + n$ there exists a top-k/w object o_i which is missing from the query k-skyband: $\exists o_i \in T_j^q \wedge o_i \notin kS_j^q$. If o_i is missing from rS_j^q, we conclude that pS_j^b did not insert it there because, according to Definition 3.8, a k-skyband does not prune objects which may become top-k/w objects in the future. From the preceding discussion, we know that PF makes two attempts to insert an object into a query k-skyband, one upon its arrival and the other just before it is dropped from the buffer. Let us first analyze the case when PF missed to insert o_i into rS_i^q in the first attempt so that o_i becomes a top-k/w object while in the buffer (i.e., $i + b > j \geq i$). We know that o_i will not be inserted in the first attempt only if $|O \overset{s}{\rhd} o_i \subset B_i \overset{s}{\rhd} o_i \subseteq W_i^q \overset{s}{\rhd} o_i| \geq k$, where O is a set of objects in pS_i^b, that is, in PF at step i. However, from the proof of Theorem 5.1 we have a contradiction $\nexists j : i + b > j \geq i \wedge o_i \in T_j^q$. Let us now analyze the case when PF missed to insert o_i into rS_{i+b}^q in the second attempt so that o_i becomes top-k/w object after being dropped from the buffer (i.e., $i+b \leq j < i+n$). As we know from the previous discussion, PF will not insert o_i in the second insertion attempt only in the following cases: (a) this object was already inserted in the first attempt and (b) this object was dominated by k or more objects from PF at step $i + b$ when being dropped from the buffer. If the object was inserted into the query k-skyband, it certainly cannot be missing since the k-skyband always keeps objects which may become top-k/w objects in the future, and thus we conclude that case (a) is impossible. For case (b), let us assume that object o_i is not inserted into rS_{i+b}^q because at step $i+b$ it holds that $|O \subset B_{i+b} \overset{s}{\rhd} o_i| \geq k$, where O is a set of objects in pS_{i+b}^b. All data objects in $O \subset B_{i+b} \overset{s}{\rhd} o_i$ are more recent than o_i and thus dropped later from the window of q. Thus $\forall l : i+n > l \geq i+b$ we have $|O \overset{s}{\blacktriangleright} o_i| = |O \overset{s}{\rhd} o_i| \geq k$, and then from Theorem 3.7 it directly follows that o_i cannot become a top-k/w object of q at any such step, that is, $l \nexists j : i + n > j \geq i + b \wedge o_i \in T_j^q$. Since from the first insertion attempt we have $\nexists j : i + b > j \geq i \wedge o_i \in T_j^q$, we conclude that $\nexists j : i + n > j \geq i \wedge o_i \in T_j^q$, which is in contradiction with our first assumption. □

Please note that it is straightforward to extend this lemma to the following variations of top-k/w processing algorithms:

—RASF (*relaxed candidate pruning algorithm with strict filter*) that uses a relaxed query k-skyband associated with a strict query filter;
—RARF (*relaxed candidate pruning algorithm with relaxed filter*) that uses a relaxed query k-skyband associated with a relaxed query filter;
—SAPF (*strict candidate pruning algorithm with probabilistic filter*) that uses a strict query k-skyband associated with a probabilistic query filter;
—SASF (*strict candidate pruning algorithm with strict filter*) that uses a strict query k-skyband associated with a strict query filter; and

—SARF (*strict candidate pruning algorithm with relaxed filter*) that uses a strict query k-skyband associated with a relaxed query filter,

where RARF and SARF use RA as the basis for the *relaxed query filter* (RF).

6. SUPPORTING MULTIPLE TOP-K/W QUERIES

This section considers a more general processing task with multiple top-k/w queries simultaneously active in the system. It requires additional data structures and some modifications when compared to a single query.

(1) We need a *query tree*, a self-balancing binary tree for storing of all active queries or query filters in the system. Therefore, when using SA and RA with filters (SF or PF), we maintain only query filters and not their queries in the query tree since each filter has a reference to its query.
(2) To store all data objects that are referenced by at least one query or filter, we use a self-balancing binary tree named *object tree*.
(3) It is possible to share a common time tree among all queries and filters in the system, which makes the implementation more efficient compared to separate time trees for all queries.
(4) We apply a single *recent object buffer* which is shared among all query filters in the system.
(5) We create an object wrapper for each object referenced by a query/filter which has references to the original object in the object tree and a corresponding query in the query tree.

In such a setting, a processing approach which does not use query indexing sequentially traverses the query tree and processes each incoming data object against the traversed query/filter. Additionally, before each object processing, we always remove dropped objects, that is, object wrappers, from the common time tree and query/filter k-skybands. Sequential processing is supported by all presented algorithms including those without filters, but is usually inefficient compared to algorithms which rely on query indexing. In the next section we explain how query indexing can be used to minimize the number of queries/filters that have to process arriving objects.

6.1. Indexing of Top-k/w Queries

Query indexing reduces the number of objects that a top-k/w query needs to process by identifying and neglecting those objects which are certainly not of interest for the query. Each algorithm defines its own query indexing threshold. If we closely analyze PA, Algorithm 1 states that a query does not need to be informed about a newly arrived object if its score is larger than the score of the last candidate object. This score is equal to the *threshold* attribute which is associated to each PA query, and we use it as the query indexing threshold. However, SA and RA do not define such an indexing threshold and need to process all incoming objects. When SA and RA are used with filters (i.e., SF, RF, or PF), the query indexing threshold is defined as the score of the top-k element in the query filter since, in both insertion attempts, only objects with scores smaller than this object are forwarded to the corresponding query. Note that all threshold values change in time. Similarly to PA, for PF we increase the threshold value (lines 13–17 in Algorithm 2) after each object removal such that it is not reset. The threshold value is increased such that to its previous value we add an absolute difference in scores of the two objects from the candidate tree with the lowest ranks. When the candidate tree becomes full again, the score of its tail becomes the new threshold value (lines 20–25 in Algorithm 1). In the case of PF we have $additional = 0$ and thus do not use additional candidate objects.

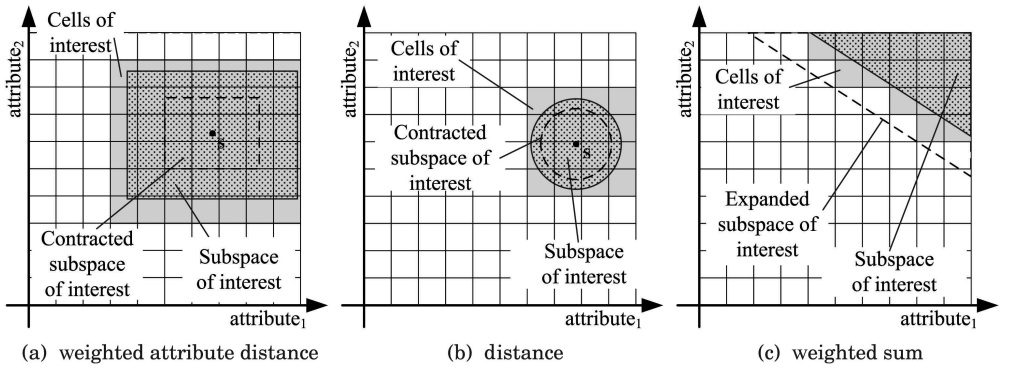


Fig. 7. Indexing of subscriptions for various scoring functions.

Query indexing techniques depend on the scoring functions associated with top-k/w queries. Hereafter we sketch indexing techniques for distance and aggregation functions to illustrate how query indexing can improve top-k/w processing performance.

As an example, Figure 7 depicts scoring functions defined in two-dimensional attribute spaces for the following functions: (a) weighted attribute distance, (b) distance, and (c) weighted sum. The *weighted attribute distance* is defined as $s(o) = c_1 \cdot |x_s - x_p|, c_2 \cdot |y_s - y_p|$, where ordered pairs (x_s, y_s) and (x_p, y_p) are points representing the query and object, respectively, while c_1 and c_2 are constants. Similarly, *a distance function* is defined as $s(o) = \sqrt{(x_s - x_p)^2 + (y_s - y_p)^2}$. In the case of a weighted sum, the scoring function is defined as $s(o) = (k_1 \cdot x_p + k_2 \cdot y_p)$.

Distance and aggregation functions process structured data objects represented as points in a multidimensional attribute space. Following the existing approaches [Mouratidis et al. 2006; Mouratidis and Papadias 2007; Böhm et al. 2007], we also use a regular grid to index queries in such spaces. A regular grid divides an attribute space in cells of equal size, while a query threshold defines the query subspace of interest as shown in Figure 7. In the case of weighted attribute distance and distance scoring functions, the query subspace of interest is defined as $s(o) < th$, while in the case of weighted sum, it is defined as $s(o) > th$, where th is the query threshold. Usually, each cell contains a list of queries whose subspace of interest completely or at least partially covers the cell. This makes the processing of incoming objects more efficient since only interested queries are informed about an object appearing in the cell. Note that the subspace of interest is covered by the *cells of interest* from the regular grid that encompass a larger portion of the attribute space than the subspace of interest itself. When the query subspace of interest changes (i.e., expands or contracts) this may change the corresponding cells of interest. However, this situation happens less often than a regular threshold change.

To our knowledge, the indexing of top-k/w queries in vector space (i.e., top-k/w queries with relevance scoring functions) is a very challenging and still open research problem which is specifically related to filtering of document streams, and is further discussed in Mouratidis and Pang [2009, 2011], Haghani et al. [2010], and Rao et al. [2014].

7. COMPLEXITY ANALYSIS

In this section we analyze the time and space complexity of SA from Böhm et al. [2007], and our RA and PA for the average- and worst-case scenarios. We assume the random-order data stream model in the average-case scenario, as is common practice in literature. In the worst-case scenario, the scores of data objects are monotonically increasing in time, and thus every incoming object has to be stored in memory as it will

become a top-k/w object at a later point in time when $n-k$ older objects are dropped from the query window. Note that the complexity analysis is performed with respect to a single top-k/w query if otherwise not explicitly stated.

7.1. Space Complexity Analysis

We express space complexity as the number of data objects referenced in memory per query. The space complexity of SA is $O(n)$ in the worst case as all data objects will eventually become top-k/w objects. In the average-case scenario, the expected number of objects in a strict query k-skyband is $O(k \cdot ln(\frac{n}{k})) = O(n')$, as shown in Zhang [2008]. Similarly, the worst-case space complexity of RA is $O(n)$, and $O(n' \cdot (1 + \frac{\gamma}{2})) = O(n')$ in the average-case scenario because the number of objects referenced by each query varies between n' and $n' + \gamma \cdot (n' - k)$, where γ is the pruning coefficient. The space complexity of PA is always $O(k + limit)$, where $limit$ is the probabilistic limit on the number of candidates.

Please note that the space complexity for a query filter (i.e., SF, RF, or PF) can be expressed analogously as for the corresponding query without a filter (i.e., SA, RA, or PA): we just need to replace n with b in the previous expressions, where b is the number of buffer objects.

The worst-case space complexity of a query k-skyband for SA and RA with filters is $O[n-(b-k)] = O(n)$ due to the fact that $b-k$ objects from the recent buffer will be kept in the query filter. In the average case, only k of b' nondominated objects in the buffer will be inserted to a query k-skyband. Thus the space complexity of a query k-skyband is $O(n' - b' + k) = O(k \cdot ln(\frac{n}{k}) - k \cdot ln(\frac{b}{k}) + k) = O(k \cdot (1 + ln(\frac{n}{b})))$ and $O(k \cdot (1+ln(\frac{n}{b})) \cdot (1+\frac{\gamma}{2}) - \frac{\gamma}{2} \cdot k) = O(k \cdot (1+ln(\frac{n}{b})))$ for SA and RA with filters, respectively. It is important to recall that, in the case when query filters are used, there is an additional overhead of b objects stored in the recent buffer. However, this buffer is shared by all queries in the system.

7.2. Time Complexity Analysis

For the time complexity analysis, we assume that u top-k/w queries with parameters k and n are processed simultaneously as they share the common time tree which influences processing performance. Furthermore, each incoming object is processed sequentially, while in each processing cycle a new data object appears and the oldest is dropped from all query windows. Note that we express time complexity as the complexity of a processing cycle per query.

SA has time complexity of $O(n + log_2(u \cdot k))$ in the worst case, where $log_2(u \cdot k)$ is due to the maintenance of the common time tree, and $O(n')$ in the average case. The time complexity of RA in the worst case is $O(log_2(n-k)^2 + log_2(u \cdot k))$ because the pruning process will never be started, while $log_2(n-k)^2$ is due to candidate tree operations. The time complexity of RA in the average case is $O(log_2((n'-k) \cdot (1+\frac{\gamma}{2}))^2) = O(log_2(n'))$, which takes into account the addition of an incoming data object to a candidate or top-k tree and object pruning every $\gamma \cdot (n'-k)$ processing cycles. For PA, in the worst case, $k+limit$ data objects will be added into the probabilistic k-skyband during n processing cycles, while on average $k + limit$ objects will be added per n incoming objects. Therefore, the time complexity is $O(1+2 \cdot \frac{k+limit}{n} \cdot log_2(u \cdot (k+limit)))$ in both cases due to the addition of an incoming object with a good rank into the time tree and removal of the lowest-ranked candidate tree object from the time tree. Note that the most time-consuming task per PA cycle is object score calculation and thus the time complexity when $k+limit \ll n$ and $u \ll n$ is approximately $O(1)$, where $\ll$ denotes "more than hundred times less than".

Note that processing with filters adds additional processing cycles and hereafter we analyze time complexity of the SA and RA with query filters. We express time complexity for a query filter maintenance by replacing n with b in the expressions for

the SA, RA, and PA without filters as follows: $O(b+log_2(u \cdot k))$ and $O(k \cdot ln(\frac{b}{k})) = O(b')$ for SF, $O(log_2(b-k)^2 + log_2(u \cdot k))$ and $O(log_2(b'))$ for RF, and $O(1)$ for PF. We get the worst-case time complexity of a query k-skyband maintenance by replacing n with $n-b+k$ in the expressions for the SA and RA without filters as $O(n-b+k+log_2(u \cdot (n-b+k)))$ and $O(log_2(n-b)^2 + log_2(u \cdot (n-b+k)))$ for the strict and relaxed k-skyband, respectively. In the average case, the number of insertions into a query k-skyband is $n \cdot \frac{k}{b}$ per window size n. In other words, an insertion happens every $\frac{k}{b}$-th cycle on average because $(n \cdot \frac{k}{b})/n = \frac{k}{b}$. From the space complexity analysis we know that the average space complexity of a query k-skyband is $O(k \cdot (1+ln(\frac{n}{b})))$ in this case. Therefore the average-case time complexity of a query k-skyband maintenance for the SA and RA with filters is $O(\frac{k}{b} \cdot k \cdot (1+ln(\frac{n}{b}))) = O(\frac{k^2}{b} \cdot (1+ln(\frac{n}{b})))$ and $O(\frac{k}{b} \cdot log_2(k \cdot (1+ln(\frac{n}{b}))))$, respectively.

8. EXPERIMENTAL EVALUATION

In this section we present an experimental study to evaluate: (1) the error rate of PA for different probabilistic criteria, (2) space consumption, and (3) processing performance of our algorithms PA, RA, RAPF, with SA and SASF defined in Böhm et al. [2007], and CPM/w and SNN from Mouratidis and Papadias [2007]. All algorithms are implemented in Java and experiments were performed in Java SE runtime environment 6u45 on a PC with 3.30 GHz Intel(R) Core(TM) i3-2120 CPU (with one core disabled). During the experiments, we allocated 2GB (out of 4GB) of available memory to the Java virtual machine.

We have selected sliding window k-NN queries as a use-case for our evaluation since they are considered as one of the most prominent top-k/w problems. Queries and data objects are represented as points in a d-dimensional attribute space. The score of an object with respect to a query is calculated as the Euclidean distance between points representing the object and query. Similarly to existing continuous k-NN monitoring approaches [Mouratidis and Papadias 2007; Böhm et al. 2007; Cheema et al. 2009], we use the regular grid as a query indexing structure.

As shown in Table VIII, one real and three synthetic datasets are used in the experimental evaluation. The real dataset is the *Lausanne urban canopy experiment* (LUCE) deployment data collected from a large-scale wireless sensor network within the project SensorScope.[7] This experiment took place on the EPFL campus from July 2006 to May 2007, and aimed at better understanding of micro-meteorology and atmospheric transport in the urban environment. It required high temporal and spatial density of measurements in order to cover heterogeneous areas of the campus. From the available data (radio duty cycle, radio transmission power, radio transmission frequency, primary buffer voltage, secondary buffer voltage, solar panel current, global current, and energy source), we extract the solar panel current, global current, primary buffer voltage, and secondary buffer voltage to create a data stream by sorting sensor readings from different sensors by their timestamps. This dataset is highly correlated in time and thus closer to the worst- than to the average-case scenario for top-k/w processing. For the synthetic datasets we randomly sampled the real dataset and generated uniform and clustered Gaussian data, and thus all of them are very close to the average-case scenario. The LUCE deployment data was preprocessed to extract four-dimensional data objects, and normalized to the values within the interval $[0, 1]$. The synthetically generated data is also within that interval. The clustered data has two randomly chosen cluster centers and variance equal to 0.1 for each dimension.

The default scenario used in all experiments is the following: We either generate or extract the set of $b + N$ data objects, and simulate the arrival of the first b objects to

[7] http://lcav.epfl.ch/files/content/sites/lcav/files/research/Sensorscope/sensorscope-monitor.zip.

Table VIII. Used Datasets

Dataset	Type	Space distribution	Time distribution
uniform	synthetic	uniform	random
clustered	synthetic	clustered	random
randomized real	synthetic	real	random
real	natural	real	correlated

Table IX. Simulation Parameters

Parameter	Symbol	Value
data stream objects	N	10^6
active queries	u	400
top objects	k	9
count-based sliding window	n	$4 \cdot 10^4$
buffer	b	2000
data dimensionality	d	2
grid resolution	ρ	10
PA: probability of error	σ	10^{-3}
PA: additional objects	a	10
RA: pruning coefficient	γ	0.2

the processing engine before starting the experiments. Note that data streams from the synthetic datasets are all random-order data streams, while a stream of objects from the real dataset is not such a stream. After that we activate u continuous queries which we randomly sample from the dataset. Next, we simulate the arrival of N objects, and finally analyze the query result streams and processor performance. The default simulation parameters used in the experiments are specified in Table IX.

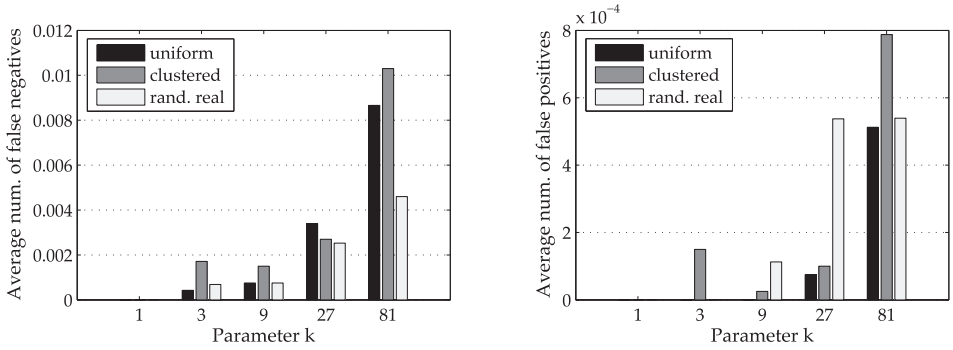
8.1. Error Rate of PA

PA produces approximate results for top-k/w queries and may erroneously report objects as potential top-k/w (false positives), or miss some objects that would become top-k/w (false negatives). Figure 8 shows the observed average number of false negative and false positive objects per subscription for three different synthetic datasets. To calculate the average values, we have run 200 iterations of the experiment. The observed error rate is 0 for all datasets when $k = 1$. For other values of parameter k, the error rate is much smaller than the theoretical upper bound which is $\sigma \cdot \frac{N}{n} = 10^{-3} \cdot \frac{10^6}{4 \cdot 10^4} = 0.25$ for the expected number of false negatives and $\frac{3}{2} \cdot \sigma \cdot \frac{N}{n} = 0.375$ for the expected number of false positives, as defined in Lemma 4.6. In the proof of this lemma we counted all potentially false positive objects as certain false positives. However, in practice many of these objects become top-k/w later on and thus the observed number of false positives is much smaller than that of false negatives.

Note that PA introduces large error rates when scores of objects are monotonically increasing in time, for example, in our real dataset which is highly correlated in time, because PA discards many incoming objects without knowing that they will become top-k/w in the future. In the opposite case when scores of incoming data objects are monotonically decreasing, PA will not generate errors since the distance scoring function prefers smaller scores.

8.2. Processing Cost

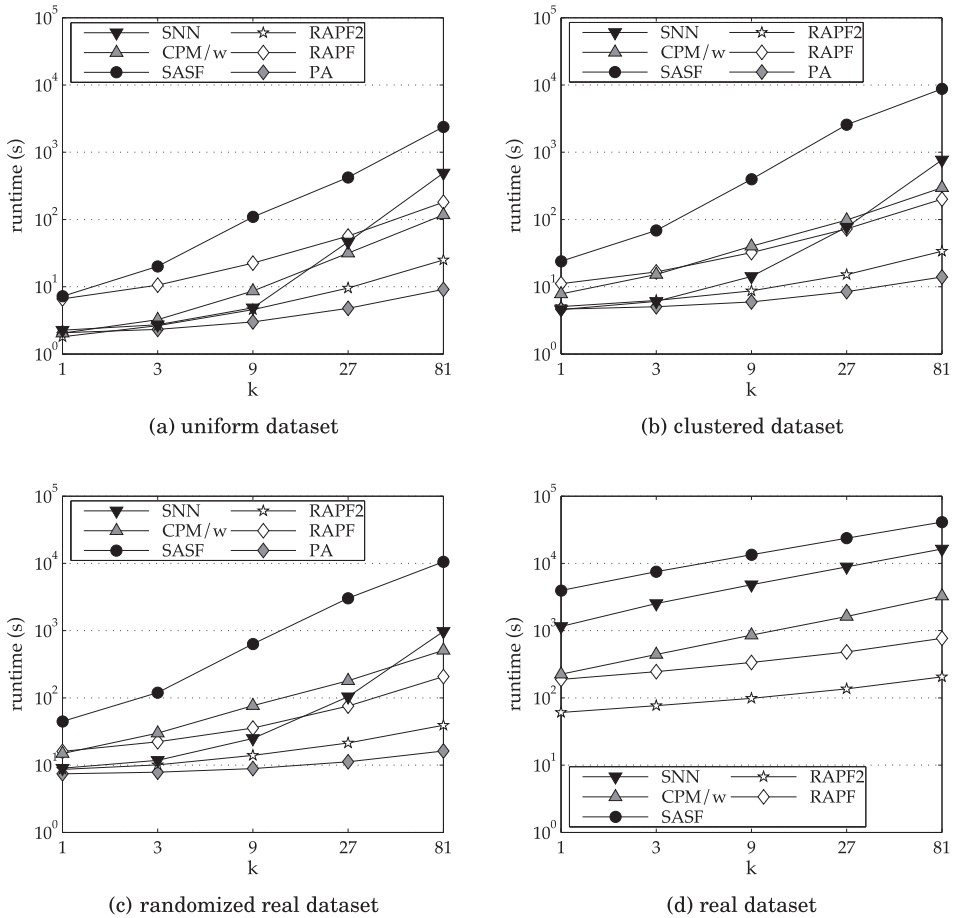
Figure 9 compares the processing cost of different algorithms expressed as simulation runtime with query indexing for the uniform, clustered, randomized real, and real



(a) average number of false negative objects per top-k/w query

(b) average number of false positive objects per top-k/w query

Fig. 8. Error rate of PA for three different synthetic datasets.



(a) uniform dataset

(b) clustered dataset

(c) randomized real dataset

(d) real dataset

Fig. 9. Processing cost for different datasets with query indexing.

dataset. We vary k in all experiments, while all other parameters are set to default values as listed in Table IX.

Note that RAPF and RAPF2 use the recent buffer of size $b = 2000$ and $b = n/2 = 20000$, respectively, where the latter buffer size corresponds to the theoretical maximum as proven in Theorem 5.1. Therefore RAPF2 presents a version of RAPF when memory is unconstrained.[8] The runtime increases in the case of the clustered dataset compared to the uniform dataset, mostly by the decrease of regular grid performance when processing spatially correlated data. In the case of synthetic datasets, we see that, for $k = 1$ and $k = 3$, algorithm PA, RAPF2, SNN, and CPM/w perform similarly. However, SNN and CPM/w scale poorly with increasing k, while RAPF, RAPF2, and especially PA scale significantly better with k. As expected, PA is faster than RAPF, and up to one order of magnitude faster than CPM/w and SNN. RAPF2 is from 4 up to 8× faster than CPM/w and SNN. Therefore RAPF2 should be used when memory is unconstrained. However, note that RAPF2 also has lower memory consumption than CPM/w and SNN which use an exhaustive indexing strategy and store all query windows objects in memory. Finally, SASF shows the worst time performance compared to other algorithms.

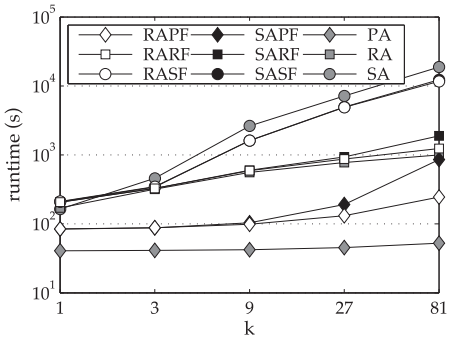
Very interesting results are observed in the case of the real dataset with stream objects that are highly correlated in time, as shown in Figure 9(d). We see that all algorithms perform much worse in this case compared to the processing of random-order data streams. CPM/w and SNN, which are designed to process mostly uniform datasets, perform much worse than our RAPF2 and RAPF for all values of parameter k. This time RAPF2 is up to one order of magnitude faster than CPM/w and up to two orders of magnitude faster than SNN. SASF is again the slowest of all examined algorithms. We do not show performance of PA since it generates a large error rate in the case of this dataset.

The series of experiments shown in Figure (10) and Figure (11) first examine algorithm performance for uniform, clustered, and randomized real datasets for different values of parameter k, both with and without query indexing, and then algorithm performance with query indexing while varying the following simulation parameters: data dimensionality d, grid resolution ρ, number of queries m, recent buffer size b, RA pruning coefficient γ, and query window size n. The default data dimensionality is $d = 4$, while all other parameters are set to their default values as listed in Table IX. We do not examine CPM/w and SNN in these experiments because (1) they cannot be used without query indexing, and (2) the goal is to examine all variations[9] of our algorithms for specific simulation parameters.

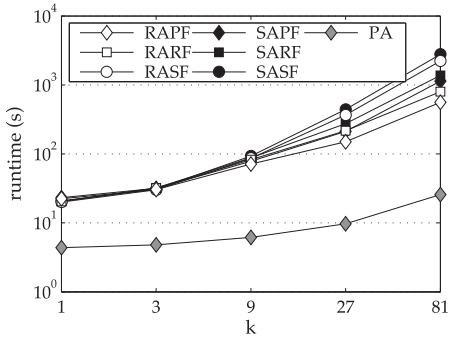
Figures 10(a), 10(c), and 10(e) show there is almost no difference in algorithm performance when various datasets are processed without query indexing. Furthermore, algorithms may be grouped according to processing performance as follows: SA, SASF, and RASF form the the first group of algorithms and offer the worst processing

[8]A question arises whether it would be beneficial to completely drop candidate pruning to get better processing performance when the memory is unconstrained. In this case, we would employ a *no candidate pruning algorithm* (NA) instead of RA for the maintenance of query and/or filter data structure (and thus would have NANF and NAPF as competitors to RAPF). However, the problem with this approach is the common time tree, as each newly arrived object has to be stored in the query/filter data structure. When we analyze the time complexity of a processing cycle per NA query, we get $O(log_2(n)^2 + log_2(u \cdot n)^2)$ in the average case since we have to add a newly appeared object to the query data structure and common time tree (as Lemma 4.8 cannot be applied to NA), and also remove the dropped object from both of these structures. This complexity is higher than the time complexity of RA which is $O(n') = O(k \cdot ln(\frac{n}{k}))$, as explained in Section 7.2, and thus it is better to prune dominated objects from the query data structure to efficiently manage the common time tree.

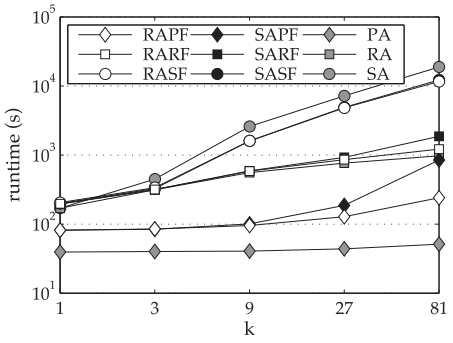
[9]Note that we also analyze RARF (RA with RF), SARF (SA with RF), and RASF (RA with SF) to show performance for all combinations of query k-skybands and filters



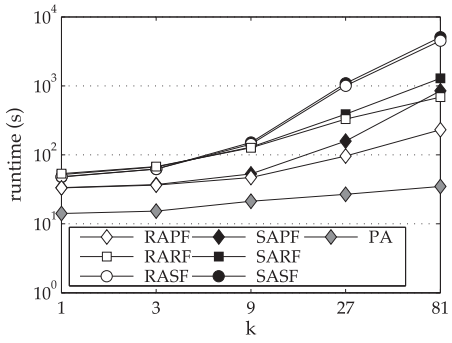
(a) uniform dataset without query indexing



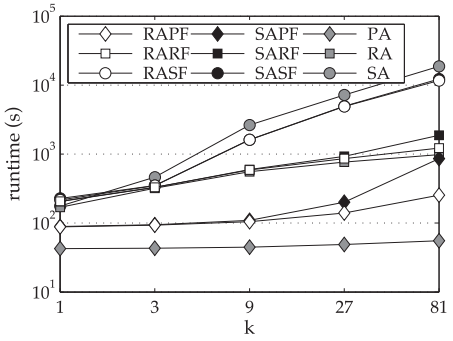
(b) uniform dataset with query indexing



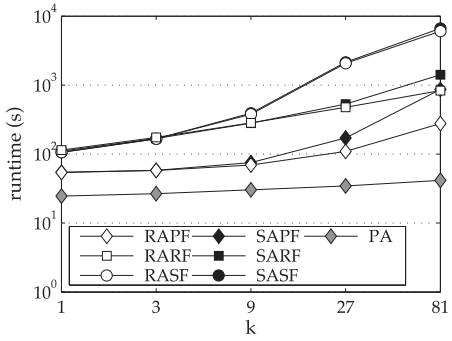
(c) clustered dataset without query indexing



(d) clustered dataset without query indexing



(e) randomized real dataset without query indexing



(f) randomized real dataset with query indexing

Fig. 10. Processing cost for different datasets and query indexing methods.

performance, especially for a large k. The second group of algorithms consists of RA, SARF, and RARF, and provides better processing performance than the first group of algorithms, however, the third group of algorithms (PA, SAPF, and RAPF) outperforms the first two groups. Clearly, the processing performance of algorithms with filters is largely influenced by the type of applied filter, as both SASF and RASF perform similarly to SA, while SARF and RARF perform similarly to RA. However, this is not the case for the third group of algorithms since PA, being the only

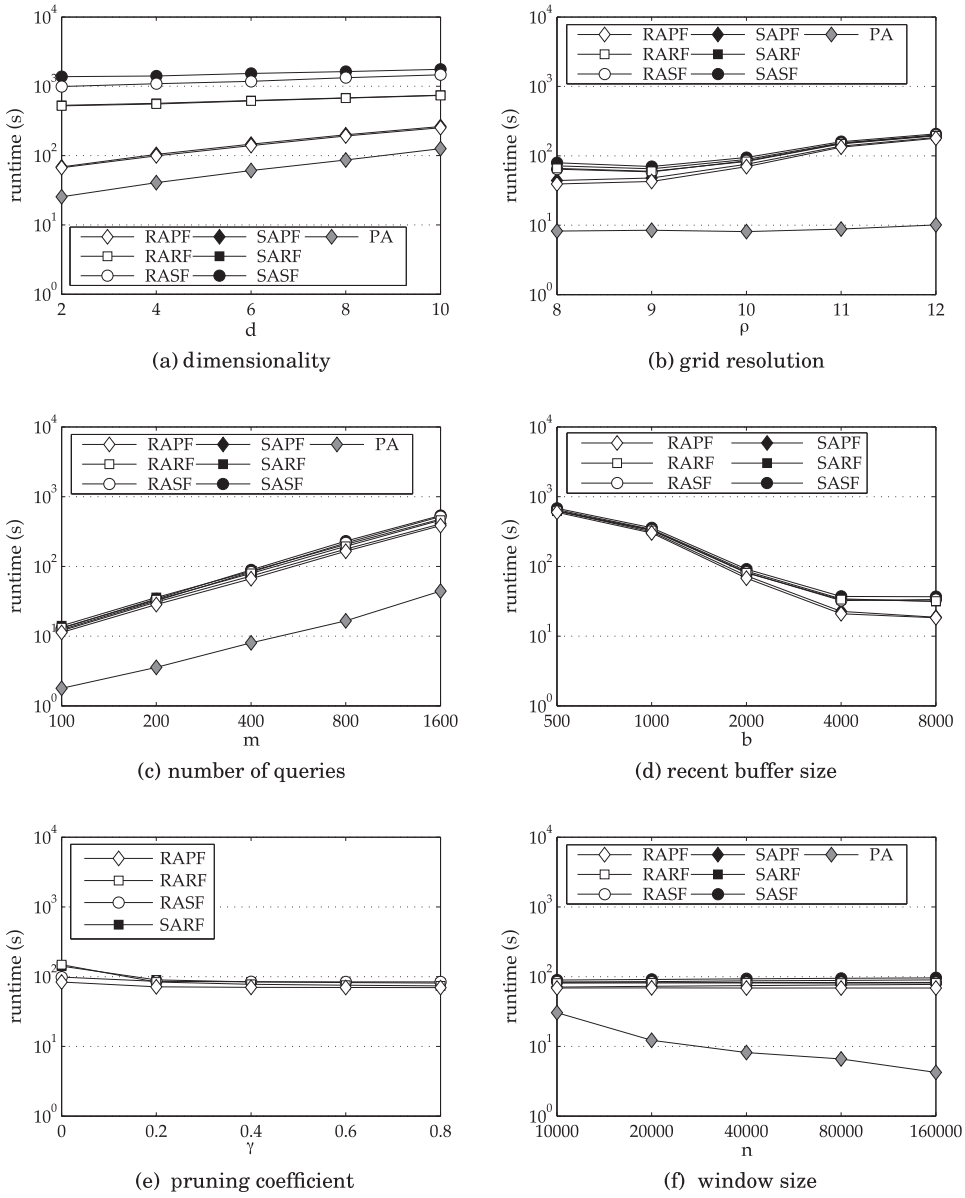


Fig. 11. Processing cost for different values of various parameters in the case of uniform dataset.

approximate algorithm, outperforms all other (exact) algorithms. More importantly, the performance of PA hardly changes when k increases, which is in accordance with the complexity analysis. We also see that SASF only slightly improves the performance of SA, while the probabilistic filter improves it almost by an order of magnitude. Note that, for large values of the parameter k, the best-performing exact algorithm is RAPF, which also scales equally or better than the competing exact algorithms.

Figures 10(b), 10(d), and 10(f) show runtime performance of different algorithms when queries are indexed in a regular grid. As expected, the regular grid offers the

best performance for the uniform dataset, while for other datasets the processing performance is only slightly improved compared to algorithms without query indexing. The explanation for this behavior is that the regular grid is adjusted for uniformly distributed data. To achieve a better processing performance in the case of nonuniform data distributions, the grid should partition the Euclidean space according to the actual data distribution. For the uniform dataset, query indexing is profitable in all situations, while for the clustered and randomized real datasets it is only profitable when the parameter k is relatively small. Additionally, note that performance of different query filters (i.e., SF, RF, and PF) is directly related to the maintenance cost of the filter k-skyband, since the query indexing threshold is defined for all filters as the score of the top-k element in the query filter (see Section 6.1).

Since the performance of the regular grid hides the algorithm performance for clustered and randomized real datasets, in Figure 11 we show the runtime only for the uniform dataset.

Figure 11(a) shows our findings when $\rho = 2$ while data dimensionality varies from 2 to 10. The runtime of PA, SAPF, and RAPF increases sublinearly when data dimensionality increases, while it scales linearly for others. It is evident that PA and algorithms with PF outperform other algorithms when processing high-dimensional data.

The results shown in Figure 11(b) analyze algorithm performance when different grid resolution ρ is used for query indexing using a regular grid. These results reveal that the minimal runtime of all algorithms is either for $\rho = 9$ or $\rho = 10$, and thus we have selected the latter resolution for our default experiential setup. Please note that, for a selected value of ρ, the grid partitions 4-dimensional Euclidean space to ρ^4 equally-sized cells.

Figure 11(c) shows that all algorithms scale linearly with increasing number of static queries u. Since grid resolution is quite high, ($\rho = 10$), the grid structure is capable of handling the increasing number of queries. Moreover, PA shows the best performance.

As we can see in Figure 11(d), a larger buffer size b results in more efficient processing, which comes at the cost of an increased memory consumption. We observe that the gain in processing performance is smaller for $b > 2000$, and therefore we have selected $b = 2000$ as the default value for buffer size in our experiments.

Smaller values of parameter γ imply that pruning of dominated objects is performed more often, while larger values of γ imply rare pruning of such objects at the cost of increased memory consumption. For $\gamma = 0$, pruning becomes strict, as in the case of SA. We have selected $\gamma = 0.2$ as the default value in our experiments since, as we can see in Figure 11(e), the gain in processing is small for larger values of γ.

Figure 11(f) demonstrates that the performance of PA improves with larger values of n, in contrast to the performance of all other algorithms that show slightly increased runtime. The reason for such an unusual finding is the increased rate of object processing for PA in case of smaller query window size because smaller n causes more frequent dropping of referenced objects and thus processing of newly arriving objects is performed more often. The runtime increases with query window size for all other algorithms, since larger n implies larger k-skyband size and therefore less efficient processing.

8.3. Space Consumption

In this simulation scenario we analyze the space consumption of the algorithms for different values of parameter k using the uniform dataset without query indexing. Similar results obtained for other random-order datasets and also with query indexing[10] are omitted due to space limitations. We also do not show space consumption of RASF,

[10]Note that PA stores additional $a = 10$ objects per top-k/w query when query indexing is employed.

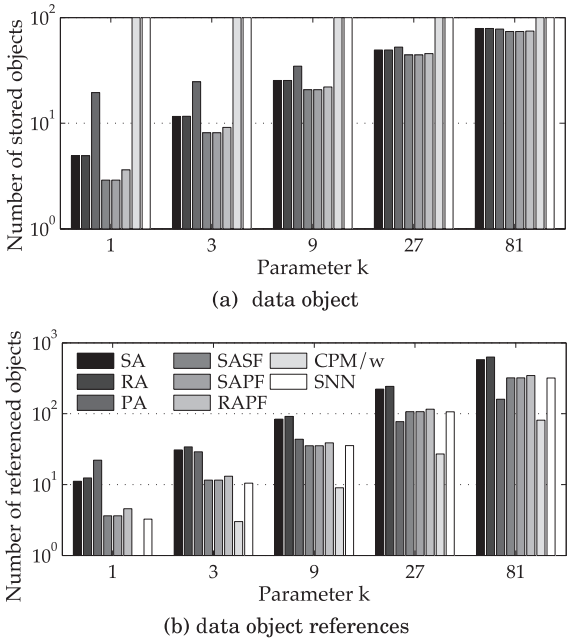


Fig. 12. Average number of data object references and stored data objects per query for the uniform dataset.

RARF, and SARF since the former two have almost identical consumptions as RAPF, while SARF's consumption is equal to SASF's. Figures 12(b) and (a) show our results expressed as the average number of data objects and data object references maintained in memory. The former is expressed as the total number of objects stored in memory divided by the number of queries u and assess to memory footprint of the algorithms, while the latter is expressed as the total number of object references for all queries divided by u, which relates to algorithm performance.

In Figure 12(a) we see that the average number of stored objects per query is similar for SA, RA, SASF, SAPF, and RAPF. PA references more objects for small values of parameter k because of the probabilistic criterion, which in this case requires the maintenance of more candidate objects than exact algorithms. Please note that CPM/w and SNN store all n objects in memory and thus always maintain $n/u = 100$ data objects in memory per query. When k increases, all algorithms asymptotically approach $n/u = 100$.

Figure 12(b) clearly shows that the exact algorithms SA and RA require a small number of object references only when parameter k is rather small, while this number grows significantly for large values of parameter k and largely impacts their runtime performance. In comparison, PA references a smaller number of objects for larger values of k ($4\times$ smaller for $k = 81$ than SA), which is beneficial for its runtime performance. In comparison to SA, RA references $\sim 10\%$ more data objects when the pruning coefficient $\gamma = 0.2$. Additionally, SA and RA reference more objects compared to their extended versions which apply query filters, while there is no significant difference in space consumption between SASF, SAPF, and SNN. As expected, RAPF references slightly more objects than SAPF or SASF. Finally, CPM/w always references only k objects per query.

9. RELATED WORK

Let us briefly discuss and compare our algorithms to related work. We classify existing approaches as either exact or approximate, and characterize them according to the

type of supported query scoring functions, assumed data stream models, and applied indexing techniques. Furthermore, we identify a set of related top-k/w problems with specific types of data streams and queries. As the algorithms introduced in Böhm et al. [2007], Mouratidis and Papadias [2007], and Mouratidis et al. [2006] are discussed in detail in Section 2, they are excluded from the discussion in this section.

Das et al. [2007] present an exact algorithm for ad-hoc top-k/w queries referencing both past and future data objects, and introduce a geometrical representation of data objects using arrangements. In contrast to our algorithms designed for ad-hoc queries referencing future data objects, this article addresses a specific problem with queries supporting linear additive aggregation scoring functions. As our solutions are generic, they could be adapted to such specific object representations and scoring functions.

A technique for the processing of e-approximate sliding window k-NN queries is presented in Koudas et al. [2004]. It partitions the attribute space with a regular grid such that the maximum distance between any pair of points in a cell is at most e, and keeps at most $K \geq k$ points per cell. Koudas et al. introduce a strategy which achieves the best accuracy with a fixed amount of memory, and the minimum memory consumption with a fixed error bound. While this approach considers the distribution of data objects in the attribute space, our PA also takes into account their distribution in time.

The first distributed solution for k-NN sliding window computation is presented in Pripužić et al. [2011]. The authors use a regular grid as a distributed indexing structure, and present specific protocols for updating query indexing thresholds between the processing nodes during real-time system operation: query activation and cancellation, and publishing of new data objects as well as node joins, departures, and failures. They assume different algorithms for k-NN sliding window processing are applied at each processing node, and evaluate the performance of such different algorithms in a distributed setting in terms of messaging overhead and system scalability. The algorithms defined and evaluated in this article which define a query indexing threshold can be used as algorithms on those processing nodes comprising the proposed distributed k-NN sliding window solution.

A technique for top-k/w processing in the case of low sliding frequencies has been recently presented in Yang et al. [2011]. The main idea behind this approach is to precompute top-k sets for future window positions, and then to update them with objects arriving in the meantime. However, this approach is very inefficient in the case of high sliding frequencies, such as when the query window slides upon each new object arrival, and thus is not comparable to our approach.

The idea of approximate processing of top-k/w queries in the context of publish/subscribe systems is introduced in our previous paper Pripužić et al. [2008], where we define a probabilistic criterion for identifying top-k and candidate objects, and use it as a useful primitive to investigate the number of objects delivered per subscription over time. Compared to the criterion defined in this article, the criterion presented in Pripužić et al. [2008] is an approximation of the probability that an object is a top-k candidate object at the time of its arrival into the processing system since it neither provides an upper bound on this probability nor provides any error guarantees.

Recently, several papers have appeared considering top-k/w queries with relevance scoring functions [Mouratidis and Pang 2009, 2011; Haghani et al. 2010; Rao et al. 2014]. While in this article we primarily focus on reducing the time complexity of top-k/w processing, these papers focus on efficient indexing of top-k/w queries in the vector space. Mouratidis and Pang [2009, 2011] propose indexing of streamed documents based on the traditional inverted file principle where, for each dictionary term, there exists an inverted list of documents from the current window and special book-keeping structure which stores current query thresholds. As shown in Haghani et al. [2010],

this approach is inefficient due to the complex query indexing and result maintenance procedures. Therefore, Haghani et al. [2010] propose storing of queries in inverted lists which group queries according to their similarity. This approach enables early stopping during the processing of incoming documents and thus significantly improves the performance. Rao et al. [2014] index queries in a covering graph instead of an inverted index, and also share evaluation results among queries to additionally improve the performance of streaming document processing.

Hereafter, we list other related top-k/w processing problems in specific application scenarios. Jin et al. [2010] design algorithms for sliding window queries over uncertain streams, where the probability of existence is assigned to each incoming data object. This is an especially challenging processing problem due to the exponential blowup in the number of possible worlds introduced by the uncertain data model. The authors propose a series of different synopses based on data compression, buffering, and exponential histograms to improve the time and space complexity of their approach. Haghani et al. [2009] deal with a problem of continuous top-k/w processing over incomplete streams where attribute values for the same data object are reported in separate nonsynchronized data streams. Their approximate algorithm is based on the correlation statistics between pairs of streams, which prunes more data objects than the exact algorithm while keeping the necessary accuracy. Nutanong et al. [2010] present an algorithm for efficient processing of moving k-NN queries which uses an incremental technique based on safe regions called the V^* diagram. This article also surveys different approaches to the problem of static top-k/w queries over moving data objects.

10. CONCLUSIONS

We study the problem of processing continuous queries that monitor top-k data objects over sliding windows (*top-k / w* queries). This problem is not trivial because data objects which are not top-k/w at the moment of their appearance in the system can later on become top-k/w objects, and thus a set of potential top-k/w objects within the query window has to be stored in memory. Existing approaches to top-k/w processing exhibit one or more of the following drawbacks: they support only a single type of scoring functions, and have either a high space or high time complexity.

In this article we define PA, the first approximate top-k/w algorithm in literature for processing random-order data streams, whose main building block is the probabilistic k-skyband, a novel data structure which maintains only those data stream objects from a sliding window that have high probability to become top-k/w objects in the future. We also introduce a novel exact top-k/w processing algorithm RAPF which uses the probabilistic k-skyband for filtering of less relevant recent data objects.

We present a comprehensive experimental evaluation which systematically compares PA and RAPF with existing algorithms in literature and reveals their characteristic properties. In all experiments, PA significantly outperforms all exact algorithms for large values of parameter k, both in terms of memory consumption and runtime performance. In particular, it has improved the simulation runtime in our experiments up to two orders of magnitude compared to the best-performing algorithms in literature (SASF, CPM/w, and SNN). It also scales better when increasing the parameter k, window size, and data dimensionality, while the observed error rate is controllable and quite low. PA also shows excellent processing performance without query indexing.

In applications when an input is not a random-order stream, it is necessary to apply exact top-k/w processing algorithms. When comparing the best-performing competing algorithms to our exact algorithms (SAPF, RA, and RAPF), one can observe the following: The best-performing algorithm is by far RAPF, which offers improved runtime performance up to one order of magnitude while requiring slightly more memory than SAPF and SNN, but less than SA or RA. Moreover, RAPF offers extremely good

runtime performance even without query indexing. Therefore one can conclude that, for top-k/w processing of random-order streams, PA is the best choice when controllable error rate is tolerable, while for processing real and correlated data streams RAPF is by far the best-performing algorithm.

APPENDIX

We define $p_{topkstep}(l, k, n, s)$ as the probability that an object with an initial rank l is a top-k/w object at step s. In this appendix, we show for which ranks l is $p_{topkstep}(l, k, n, s)$ increasing and convex as a discrete function of step s.

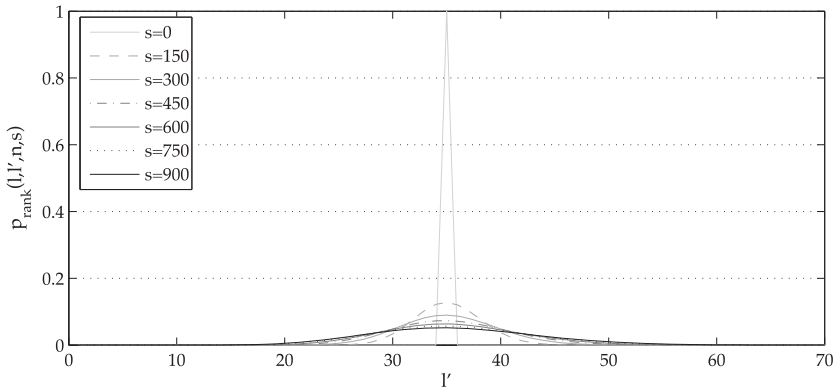
From Lemma 4.3 we know that $p_{rank}(l, l', n, s) = \sum_{y=0}^{l'-1}[p_{drop}(l, n, s, l - l' + y) \cdot p_{arrival}(l, n, s, y)]$ is the probability that an object with initial rank l will have rank l' at step s. Figure 13 shows $p_{rank}(l, l', n, s)$ for $l = 35$ and $n = 1000$, and similar figures can be drawn for any combination of l and n. In Figure 13(a) we see that as s increases the curve becomes flatter and wider. This is expected as at step $s = 0$ each object is always at its initial rank l, but later on as time passes it has higher probability to acquire other ranks. Additionally, Figure 13(a) also shows that the probability of initial rank $l' = 35$ is continuously decreasing in time, while for ranks which are close to the initial rank but higher than it (e.g., $l' = 34$), these probabilities are first increasing, and then decreasing in time. Furthermore, for ranks which are even higher than the former (e.g., rank $l' = 27$), these probabilities are continuously increasing in time. In Figure 13(b) we observe that the increase rate of these probabilities for rank $l' = 27$ is higher for earlier steps s than for later steps. Finally, for the highest ranks (e.g., rank $l' = 16$ shown in Figure 13(c)) the increase rate of these probabilities is rising continuously. We are interested in these ranks since their probabilities (of becoming a top-k/w object at step s) are increasing and convex as a discrete function of step s.

The first of the following lemmas proves the criterion which is valid for initial ranks with increasing probabilities, while the second lemma introduces an additional criterion which is valid only for initial ranks whose probabilities are both increasing and convex.

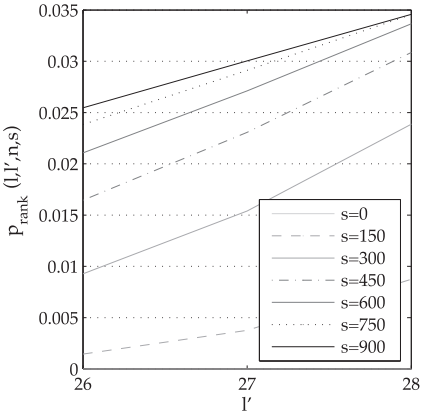
LEMMA A.1. *For an object with initial rank l, its probabilities $p_{topkstep}(l, k, n, s)$ of being a top-k/w object at step s increase with the value of step s if the following holds.*

$$l > \frac{n - 2 \cdot k + 2 \cdot k \cdot n + 1}{2 \cdot n} + \frac{\sqrt{-8 \cdot k^2 \cdot n + 4 \cdot k^2 + 8 \cdot k \cdot n^2 + 4 \cdot k \cdot n - 4 \cdot k - 7 \cdot n^2 + 2 \cdot n + 1}}{2 \cdot n} \tag{12}$$

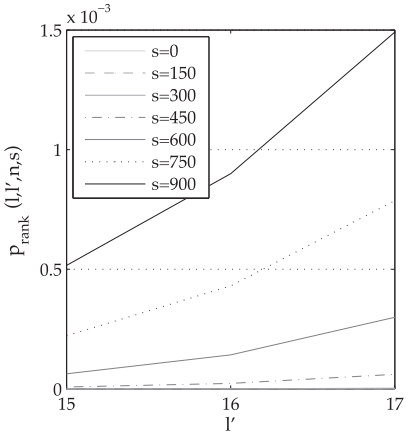
PROOF. From the preceding discussion we have to find ranks l for which holds it $p_{topkstep}(l, k, n, n-1) > p_{topkstep}(l, k, n, n-2)$, where from Lemma 4.3 we have that $p_{topkstep}(l, k, n, s) = \sum_{l'=1}^{k} p_{rank}(l, l', n, s) = \sum_{l'=1}^{k} \sum_{y=0}^{l'-1}[p_{drop}(l, n, s, l - l' + y) \cdot p_{arrival}(l, n, s, y)]$. Moreover, as higher top-k ranks ($l' < k$) are more to the left than rank $l' = k$, it is obvious that the previous inequality holds if $p_{rank}(l, k, n, n-1) > p_{rank}(l, k, n, n-2)$ is true. Therefore we have to prove that $p_{arrival}(l, n, n-1, k-1) \cdot p_{drop}(l, n, n-1, l-1) > p_{drop}(l, n, n-2, l-1) \cdot p_{arrival}(l, n, n-2, k-1) + p_{drop}(l, n, n-2, l-2) \cdot p_{arrival}(l, n, n-2, k-2)$. From Eqs. (5) and (6) we then have $\frac{n}{2 \cdot n-1} \cdot \frac{\binom{l+k-2}{k-1}\binom{2 \cdot n-l-k}{n-k}}{\binom{2 \cdot n-2}{n-1}} \cdot 1 > \frac{\binom{l-1}{l-1}\binom{n-l}{n-l-1}}{\binom{n-1}{n-2}} \cdot \frac{n}{2 \cdot n-2} \cdot \frac{\binom{l+k-2}{k-1}\binom{2 \cdot n-l-k-1}{n-k-1}}{\binom{2 \cdot n-3}{n-2}} + \frac{\binom{l-1}{l-2}\binom{n-l}{n-l}}{\binom{n-1}{n-2}} \cdot \frac{n}{2 \cdot n-2} \cdot \frac{\binom{l+k-3}{k-2}\binom{2 \cdot n-l-k}{n-k}}{\binom{2 \cdot n-3}{n-2}}$. By simplifying this inequality we get $\frac{1}{2 \cdot n-1} > \frac{1}{(n-1)^2} \cdot [(n-l) \cdot \frac{n-k}{2 \cdot n-l-k} + (l-1) \cdot \frac{k-1}{l+k-2}]$. By solving the quadratic inequality we get two roots: $r_1 = \frac{n-2 \cdot k+2 \cdot k \cdot n+1+\sqrt{-8 \cdot k^2 \cdot n+4 \cdot k^2+8 \cdot k \cdot n^2+4 \cdot k \cdot n-4 \cdot k-7 \cdot n^2+2 \cdot n+1}}{2 \cdot n}$ and



(a) positions from 0 to 70



(b) zoomed ranks from 26 to 28



(c) zoomed ranks from 15 to 17

Fig. 13. Probability that an object with initial rank $l = 35$ has rank l' at step s when $n = 1000$.

$r_2 = \frac{n-2\cdot k+2\cdot k\cdot n+1-\sqrt{-8\cdot k^2\cdot n+4\cdot k^2+8\cdot k\cdot n^2+4\cdot k\cdot n-4\cdot k-7\cdot n^2+2\cdot n+1}}{2\cdot n}$. Finally, since it is straightforward to prove that $r_2 < k$, we conclude r_1 is the only valid solution. □

LEMMA A.2. *For an object with initial rank l, its probabilities* $p_{topkstep}(l,k,n,s)$ *of being a top-k/w object at step s are both increasing and convex with the value of step s if the following holds.*

$$l > \frac{3\cdot n-4\cdot k+2\cdot k\cdot n+3}{2\cdot n+2} + \frac{\sqrt{3\cdot(-8\cdot k^2\cdot n+4\cdot k^2+8\cdot k\cdot n^2+4\cdot k\cdot n-4\cdot k-5\cdot n^2-2\cdot n+3)}}{2\cdot n+2} \tag{13}$$

PROOF. From the earlier discussion we have to find ranks l for which it holds that $p_{topkstep}(l,k,n,n-1) - p_{topkstep}(l,k,n,n-2) > \frac{1}{n-1}\cdot p_{topkstep}(l,k,n,n-1)$, where again from Lemma 4.3 we have that $p_{topkstep}(l,k,n,s) = \sum_{l'=1}^{k} p_{rank}(l,l',n,s) = \sum_{l'=1}^{k}\sum_{y=0}^{l'-1}[p_{drop}(l,n,s,l-l'+y)\cdot p_{arrival}(l,n,s,y)]$. Moreover, as higher top-k ranks ($l' < k$) are more to the left than rank $l' = k$, it is obvious that the previous

Table X. Highest Rank l for which Probabilities $p_{topkstep}(l, k, n, s)$ of Being a Top-k/w Object at Step s Increase with the Value of Step s

$n \setminus k$	1	2	5	10	20	50	100	200	500
10^3	4	7	12	19	33	69	125	232	539
10^4	4	7	12	19	33	69	126	236	555
10^5	4	7	12	19	33	69	126	237	557
10^6	4	7	12	19	33	69	126	237	557

inequality holds if $p_{rank}(l,k,n,n-1) - p_{rank}(l,k,n,n-2) > \frac{1}{n-1} \cdot p_{rank}(l,k,n,n-1)$ is true. Therefore we have to prove that $p_{arrival}(l,n,n-1,k-1) \cdot p_{drop}(l,n,n-1,l-1) - p_{drop}(l,n,n-2,l-1) \cdot p_{arrival}(l,n,n-2,k-1) - p_{drop}(l,n,n-2,l-2) \cdot p_{arrival}(l,n,n-2,k-2) > \frac{1}{n-1} \cdot p_{arrival}(l,n,n-1,k-1)$. From Eqs. (5) and (6) we then have $\frac{n}{2 \cdot n-1} \cdot \frac{\binom{l+k-2}{k-1}\binom{2 \cdot n-l-k}{n-k}}{\binom{2 \cdot n-2}{n-1}} \cdot 1 - \frac{\binom{l-1}{l-1}\binom{n-l}{n-l-1}}{\binom{n-1}{n-2}} \cdot \frac{n}{2 \cdot n-2} \cdot \frac{\binom{l+k-2}{k-1}\binom{2 \cdot n-l-k-1}{n-k-1}}{\binom{2 \cdot n-3}{n-2}} - \frac{\binom{l-1}{l-2}\binom{n-l}{n-l}}{\binom{n-1}{n-2}} \cdot \frac{n}{2 \cdot n-2} \cdot \frac{\binom{l+k-3}{k-2}\binom{2 \cdot n-l-k}{n-k}}{\binom{2 \cdot n-3}{n-2}} > \frac{1}{n-1} \frac{n}{2 \cdot n-1} \cdot \frac{\binom{l+k-2}{k-1}\binom{2 \cdot n-l-k}{n-k}}{\binom{2 \cdot n-2}{n-1}}$. By simplifying this inequality we get $\frac{1}{2 \cdot n-1} - \frac{1}{(n-1)^2} \cdot [(n-l) \cdot \frac{n-k}{2 \cdot n-l-k} + (l-1) \cdot \frac{k-1}{l+k-2}] > \frac{1}{n-1} \cdot \frac{1}{2 \cdot n-1}$. By solving the quadratic inequality we get two roots: $r_1 = \frac{3 \cdot n-4 \cdot k+2 \cdot k \cdot n+3+\sqrt{3 \cdot (-8 \cdot k^2 \cdot n+4 \cdot k^2+8 \cdot k \cdot n^2+4 \cdot k \cdot n-4 \cdot k-5 \cdot n^2-2 \cdot n+3)}}{2 \cdot n+2}$ and $r_2 = \frac{3 \cdot n-4 \cdot k+2 \cdot k \cdot n+3-\sqrt{3 \cdot (-8 \cdot k^2 \cdot n+4 \cdot k^2+8 \cdot k \cdot n^2+4 \cdot k \cdot n-4 \cdot k-5 \cdot n^2-2 \cdot n+3)}}{2 \cdot n+2}$. Finally, since it is straightforward to prove that $r_2 < k$, we conclude r_1 is the only valid solution. □

In Table X we use Lemma A.1 (and Lemma A.2) to show, for typical values of parameters k and window sizes n, those ranks for which probabilities $p_{topkstep}(l,k,n,s)$ are increasing in time. If we compare Table X with Table V, we see that probabilities $p_{topkstep}(l,k,n,s)$ increase in time and are convex for the last candidate in each of probabilistic k-skyband, which demonstrates the correctness of our approach in Section 4.1.1. Note that values in Table X stop rising with increasing value of window size n, and very slowly rise with increasing value of parameter k.

REFERENCES

Christian Bohm, Beng Chin Ooi, Claudia Plant, and Ying Yan. 2007. Efficiently processing continuous kNN queries on data streams. In *Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE'07)*. 156–165.

Amit Chakrabarti, Graham Cormode, and Andrew Mcgregor. 2008a. Robust lower bounds for communication and stream computation. In *Proceedings of the 40th ACM Annual Symposium on Theory of Computing (STOC'08)*. ACM Press, New York, 641–650.

Amit Chakrabarti, T. S. Jayram, and Mihai Patrascu. 2008b. Tight lower bounds for selection in randomly ordered streams. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'08)*. 720–729.

Sharma Chakravarthy and Qingchun Jiang. 2009. *Stream Data Processing: A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing*. 1st Ed. Springer.

Muhammad Aamir Cheema, Xuemin Lin, Ying Zhang, Wei Wang, and Wenjie Zhang. 2009. Lazy updates: An efficient technique to continuously monitoring reverse kNN. *Proc. VLDB Endow.* 2, 1, 1138–1149.

Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Nikos Sarkas. 2007. Ad-hoc top-k query answering for data streams. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07)*. 183–194.

Lukasz Golab and M. Tamer Ozsu. 2003. Issues in data stream management. *SIGMOD Rec.* 32, 2, 5–14.

Sudipto Guha and Andrew Mcgregor. 2006. Approximate quantiles and the order of the stream. In *Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'06)*. ACM Press, New York, 273–279.

Parisa Haghani, Sebastian Michel, and Karl Aberer. 2009. Evaluating top-k queries over incomplete data streams. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM'09)*. ACM Press, New York, 877–886.

Parisa Haghani, Sebastian Michel, and Karl Aberer. 2010. The gist of everything new: Personalized top-k processing over Web 2.0 streams. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM'10)*. ACM Press, New York, 489–498.

Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.* 40, 4.

Cheqing Jin, Ke Yi, Lei Chen, Jeffrey Xu Yu, and Xuemin Lin. 2010. Sliding-window top-k queries on uncertain streams. *VLDB J.* 19, 3, 411–435.

Nick Koudas, Beng Chin Ooi, Kian-Lee Tan, and Rui Zhang. 2004. Approximate NN queries on streams with guaranteed error/performance bounds. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB'04)*. 804–815.

Andrew McGregor. 2008. Recent results on processing random-order streams and space-efficient sampling. In *Proceedings of the 46th Annual Allerton Conference on Communication, Control, and Computing (ALLERTON'08)*. 206–208.

Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. 2006. Continuous monitoring of top-k queries over sliding windows. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'06)*. ACM Press, New York, 635–646.

Kyriakos Mouratidis and Hweehwa Pang. 2009. An incremental threshold method for continuous text search queries. In *Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE'09)*. 1187–1190.

Kyriakos Mouratidis and Hweehwa Pang. 2011. Efficient evaluation of continuous text search queries. *IEEE Trans. Knowl. Data Engin.* 23, 10, 1469–1482.

Kyriakos Mouratidis and Dimitris Papadias. 2007. Continuous nearest neighbor queries over sliding windows. *IEEE Trans. Knowl. Data Engin.* 19, 6, 789–803.

J. Ian Munro and Mike S. Paterson. 1978. Selection and sorting with limited storage. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science (SFCS'78)*. 253–258.

Sarana Nutanong, Rui Zhang, Egemen Tanin, and Lars Kulik. 2010. Analysis and evaluation of V*-kNN: An efficient algorithm for moving kNN queries. *VLDB J. 19,* 3, 307–332.

Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive skyline computation in database systems. *ACM Trans. Database Syst.* 30, 1, 41–82.

Kresimir Pripuzic, Ivana Podnar Zarko, and Karl Aberer. 2011. Distributed processing of continuous sliding-window k-NN queries for data stream filtering. *World Wide Web* 14, 5–6, 465–494.

Kresimir Pripuzic, Ivana Podnar Zarko, and Karl Aberer. 2008. Top-k/w publish/subscribe: Finding k most relevant publications in sliding time window w. In *Proceedings of the 2nd International Conference on Distributed Event-based Systems (DEBS'08)*. ACM Press, New York, 127–138.

Weixiong Rao, Lei Chen, Shudong Chen, and Sasu Tarkoma. 2014. Evaluating continuous top-k queries over document streams. *World Wide Web* 17, 1, 59–83.

Panayiotis Tsaparas, Themistoklis Palpanas, Yannis Kotidis, Nick Koudas, and Divesh Srivastava. 2003. Ranked join indices. In *Proceedings of the 19th International Conference on Data Engineering (ICDE'03)*. 277–288.

Di Yang, Avani Shastri, Elke A. Rundensteiner, and Matthew O. Ward. 2011. An optimal strategy for monitoring top-k queries in streaming windows. In *Proceedings of the 14th International Conference on Extending Database Technology (EDBT'11)*. ACM Press, New York, 57–68.

Ying Zhang. 2008. Computing order statistics over data streams. Ph.D. dissertation, University of New South Wales, Sydney, Australia.

Received July 2013; revised June 2014; accepted July 2014