Efficient Top-K Continuous Query Processing Over Sliding Window Model (SWM) Method on Uncertain Data Stream

RAJA AZHAN SYAH RAJA WAHAB, SITI NURULAIN MOHD RUM, HAMIDAH IBRAHIM, ISKANDAR ISHAK
Faculty of Computer Sciences & Information Technology,
Universiti Putra Malaysia,
43400 UPM Serdang, Selangor Darul Ehsan,
MALAYSIA

Abstract: - Query processing using the Uncertain Data Stream (UDS) can be complex in many technological scenarios due to inconsistencies, unclear information, and interpretation latency. As a result of both the sheer amount of data generated and the rate of change, traditional processing methods are in dire need of an upgrade. UDS consists of a finite set of states known as possible worlds (PW), and enhancing data organization can lead to more accurate extraction of user preferences. The number of possible world instances in UDS grows exponentially, making achieving Top-k query processing quickly a significant challenge. Different methods are available to handle Top-k queries in various types of UDS, and their key concerns include reducing duplicate scans of the entire dataset, enhancing uncertainty computation, and focusing on processing the latest tuple item entry. It appears that there have been limited studies conducted on the issue of UDS using the Sliding Window Model (*SWM*). The current approach for handling continuous queries on UDS within the *SWM* has proven to be ineffective, resulting in complex trade-offs between maximizing probability and generating high-scoring result sets. The challenge is to find the correct result list that satisfies a Top-k query predicate with scoring and probability. This study proposes a framework for processing Top-k queries for *UDS* using the sliding window model to improve efficiency. The study also discusses an improved optimization method for reducing computational redundancy in the context of the sliding window model and Top-k query processing. Overall, this research will significantly contribute to the Top-k computational query processing field.

Key-Words: - Top-k, Uncertain Data Stream (UDS), Sliding Window Model (SWM), Tuple Items, Possible Worlds (PW), Query Processing, Bucket Set, Computation, Segmentation, Optimization.

Received: April 13, 2024. Revised: August 16, 2024. Accepted: September 9, 2024. Published: October 29, 2024.

1 Introduction

Conventional query processing methods utilizing in-memory algorithms need help to handle the extensive data stream volumes. Therefore, creating efficient techniques for processing Top-k queries and integrating a reliable Data Stream Management System (*DSMS*) to facilitate query processing from various data sources is crucial. The transition from a centralized to a distributed data environment is critical in ranked retrieval, [1]. A user-defined scoring function and a specific query generate k-tuple items with the highest scores, [2]. This method is highly important in a range of emerging applications. These include object tracking, RFID technology, sensor networks, information extraction, and data integration, [3]. Probability distributions are frequently used to describe situations with uncertainty in data values rather than predictability, [4].

Processing *UDS* can be quite challenging due to a few factors, including the real-time generation of tuple items, the lack of control over their arrival order, the unlimited scale of data streams, and the discarding of processed data stream objects, [5]. Overestimating the required window size could cause unexpected and undesired tuple item returns, [6]. *UDS* is considered uncertain when it contains an uncertain object model, a possible world semantic model, or both. Several studies have been conducted to create models that describe *UDS* in semantics. These models can represent relationships in the form of sequences of events [7], including semi-structured data models [8], [9], stream data models [10], relational data models [11], [12], and multidimensional data models [12], among others. The main focus of *UDS* research is possible instances in the world, which are represented by possible worlds, [13], [14]. The Sliding Window Model (*SWM*) represents all possible world settings

by generating tuples in combinations. Within this model, multiple potential worlds can be generated at a specific timestamp, which leads to a significant increase in the number of tuples as the size of the sliding window expands, [15].

Processing continuous Top-k queries over a *SWM* is a complex problem when dealing with uncertain data streams, [16], [17], [18]. To maintain an up-to-date *SWM*, it is crucial to continuously process sliding window queries and alert users of any changes in query results, [19]. Therefore, this study focuses on exploring the best combination of tuple items that satisfy the scoring requirements, probabilities, or both for the Top-k query.

After this introduction, the paper is organized as follows. Section 2 reviews related work that serves as a basis for identifying gaps in the research. Section 3 highlights the study's contribution, followed by Section 4 presents a preliminary problem statement for better understanding. The methodology is detailed in Section 5, including algorithms and examples. Section 6 presents the results of the experiment. Finally, in Section 7, we will provide concluding remarks for this study.

2 Related Work

This study delves into the essential principles of continuous query preferences in a centralized environment. It focuses on various types of continuous queries such as top-k, skyline, and top-k dominating that DSMS executes with stream inputs limited by sliding windows. In the literature, many techniques have been proposed based on top-k queries, [20], [21], [22], [23], [24], [25], [26], [27], [28] that researchers have explored in various areas. These areas include both centralized environments with *SWM* [29], [30], [31], [31], [32], [33], [34], Dominant Relationship Analysis (*DRA*) [35], without *SWM* [36], [37], [38], *UFIM* with *UFIMTopK* [39], and Top-k query over an Incomplete Data Stream (*Topk-iDS*) [40], [41], [42], [43] approach. The complexity arises from the need to aggregate scores of candidate items and their probabilities in *UDS*, [43].

2.1 Top-k Queries

To define various top-k ranking query semantics, different parameters can be adapted, such as *UTopk* [44], *PT-k* [44], *PTk-S* [44], *eScore Rank* [44], *Global Topk* [44], *UTR* [44], *PTD* [45], among others. *UTopk* focuses on probabilistic threshold top-k queries. It utilizes user-provided probability thresholds to filter and rank data items. *PT-k* analyzes large datasets with significant uncertainty, making it easier to derive top-k results based on user-defined scores. Significant findings were made regarding top-k best probability queries. This study emphasizes the selection of probabilistic tuples with the highest top-k scores and probabilities. The algorithm for selecting the top-k best probabilities demonstrates superior speed and efficiency compared to the Probability Threshold Technique (*PT-k*). In study proposed by *DRA*, it minimizes the number of tuple elements needed for query processing. By adopting this approach, they were able to decrease the number of tuple items that require processing and limit the generation of potential world instances.

2.2 Sliding Window Model (*SWM*)

In response to continuous and uncertain user requests, this study delves into the topic of UDS and proposes methods for computing top-k on tuple items. A variety of tuple item combinations are generated by the SWM, which characterizes the probable world context, [46]. Tuple items can be generated in the SWM using a timestamp and several possible worlds. However, the number of tuple elements grows exponentially with the size of the sliding window frame. It takes a lot of time and energy to handle incoming and outgoing tuple elements in rapid streams effectively. Handling queries consisting of the most probable top-k tuple query sets is the most difficult attempt to handle, [46]. The method uses probabilities and scores to select the top k tuples within each sliding window, regardless of the number of tuple items.

2.3 Tuple-Level Uncertainty (*TLU*)

In 2018, a novel method was introduced for addressing preference queries. They developed two algorithms, *UFIM* and *UFIMTopK*, to efficiently detect frequent item sets from uncertain data streams based on thresholds and ranks. To facilitate efficient top-k queries, the SAP method employs a partitioning mechanism. Finding and keeping a selected set of tuple items in the window frame is essential for getting answers when the window changes. To decrease the re-scanning interval even further, one could reduce the re-scanning frequency in the sliding window, mainly when high-scoring tuple items are located within the window frame. *Topk-iDS* algorithm that can determine the top-k tuple items with the highest-ranking scores from an incomplete data stream. Their algorithm uses a sliding window framework that combines count-based and time-based SWM to monitor the highest-k tuple elements. To address issues like insufficient information and uncertainty, their work suggests

using cost modeling that can involve creating dynamic data summaries and pruning techniques to effectively reduce the search space for *Topk-iDS*.

2.4 Comparative Analysis

Table 1 provides a comparative analysis of various methodologies in continuous Top-k query processing over uncertain data streams. It highlights the key features, advantages, and limitations of each approach.

Table 1. An Analysis and Comparison of Algorithms

Method	Exact-k	Containment	Unique Ranking	Stability	Invariance	Faithfulness
Global Topk	Weak	Fail	Satisfied	Satisfied	Satisfied	Fail
U-Topk	Fail	Fail	Satisfied	Satisfied	Satisfied	Fail
K-best ranking with PT-k	Weak	Weak	Satisfied	Satisfied	Satisfied	Weak
UFIM and *UFIMTopK*	Weak	Satisfied	Satisfied	Weak	Satisfied	Weak
DRA with Pk-Topk	Weak	Satisfied	Satisfied	Satisfied	Weak	Fail
Topk-iDS	Weak	Satisfied	Satisfied	Satisfied	Satisfied	Weak
Proposed *SWMTopk-kDelta*	Satisfied	Satisfied	Satisfied	Satisfied	Satisfied	Satisfied

Most academics have concentrated on top-k query processing for specific data types and their modifications, but when *UDS* is involved, processing with the sliding window model approach is not enabled. This study utilized the following categories to structure the prior research: (i) Operates on the concept that probability and top-k score methods are frequently employed to characterize *UDS* tuple items. (ii) Processing streaming data poses numerous challenges, and this study seeks to address issues arising from *UDS* by creating alternative scenarios. Representing *UDS* using a possible world model can be a challenging task.

3 Contributions

This study employs a *SWM* to investigate and compute the top-k query on uncertain data streams. To achieve the intended contribution, it is vital to thoroughly analyze various considerations before implementing the proposed framework:

- The *UDS* model requires further examination and analysis of its categories.
- The proposed SWM needs to be evaluated for its efficacy in handling continuous queries.
- To obtain the appropriate Top-k results based on scoring and probability, query processing is utilized.
- There's a need for optimization to decrease the computation time and complexity of Top-k results efficiently.

To improve the computational efficiency of managing tuple item scores and probabilities, examining the critical components of a suitable top-k query method for uncertain data streams is essential. Our proposed approach provides a systematic process for achieving high scores and maximum probabilities across all possible world situations. This study introduces a new problem in processing continuous queries that aim to find the top-k tuple items with various fundamental characteristics. It is crucial to optimize strategies for top-k over SWM processing, considering the contributions highlighted before. An empirical study was conducted on both real-world and synthetic datasets to validate the effectiveness of these techniques.

Therefore, this paper is crucial in top-k query processing as it provides comprehensive knowledge, ensuring efficient and accurate access to information and leading to overall user satisfaction. With the exponential growth of data, effective query processing becomes even more critical. By improving the retrieval process, managing complex queries, optimizing resources, enhancing user experience, supporting advanced features, and ensuring data integration, query processing plays a significant role in the success and effectiveness of information retrieval systems. As data volumes grow, robust top-k query processing mechanisms will only increase, making it a critical area of focus for researchers and practitioners.

4 Preliminary and Problem Statement

4.1 Preliminary

[47], analyzed three models for managing uncertain data: fuzzy, evidence-oriented, and probabilistic methods. Our study utilizes the completed model to depict the specific example and the Uncertain Data Model (*UDM*) derived from *Definition 1*. This model will be called the Sliding Uncertain Data Model (*SUDM*). The model is characterized by its inclusion of Uncertainty (ALU) and Tuple Level Uncertainty (TLU).

*Definition 1 (**Uncertain Data Stream**, **UDS**)*: A subscript denotes the point in time at which the tuple item arrives. A tuple item s_i is a point with actual values in d dimensions that are not overly specific. A suitable vector $S[q] = \{s_{q-L+1}, \ldots s_{q-1}, s_q\}$ for a

subsequence of stream S defined as $S = \{s_1, s_2, ..., s_i...\}$ and the Top-k query seeks for a record vector of size L. The starting point for indexing a tuple on the uncertain data stream and subsequent application of the sliding window model is denoted by N, and it is given a series of tuples s_i, where $0 \leq i \leq n$.

Table 2 presents a sample of *UDS* that contains 10 tuple items. Each item has 3 attributes, which include a probability value. In this work, we adopt the baseline sliding window semantics, it is defined as follows:

Table 2. Example of an Uncertain Data Stream Set

Timestamp (secs)	t_{id}	Tuple Item	Score	Probability
5	t_1	R1	80	0.3
10	t_2	R2	65	0.4
10	t_3	R3	45	0.5
15	t_4	R4	30	1
20	t_5	R5	50	0.8
20	t_6	R6	25	0.2
25	t_7	R7	40	0.5
26	t_8	R8	55	0.6
26	t_9	R9	78	0.4
30	t_{10}	R10	90	0.8

*Definition 2 (**Sliding Window Model Semantics**)*: For a given $S = \{s_1, s_2, ..., s_i...\}$, a windowed stream operator operates on incoming tuples using the sliding window, $SW(S, Win)$, where Win represents the window size of $|SW|$. Due to the existential uncertainty of specific tuples in S, the borders of the SWM may also appear uncertain. Subsequently, the most recent objects generated will become invalid after $|SW|$ time instances. The set of *UDS* tuple items within the current sliding window at time t is represented as $SW\ [t-|SW|+1, t]$. This is indicated by $Prob(|SW(S, win)|= win) < 1$. This approach follows *delta's* proposed sliding window semantics (attribute, delta) and the uncertainty-independent model for possible world semantics.

It is necessary to group tuple items and adjust the synopsis based on density probability with a running timestamp. Candidates are retrieved using a selection technique outlined below:

*Definition 3 (**Group Membership, GM):*** The tuple items within a window are arranged into groups based on their attribute values in the grouping list $L = \{A_i, A_j, ..., A_n\}$, similar to the function of the corresponding operation in extended relational algebra (such as *COUNT*, *SUM*, *MAX*, or *AVG*).

Once the group membership strategy is implemented and all tuple items are correctly projected, the partitioning process on *SWM* can be defined as follows:

*Definition 4 (**Sliding Window Partition,** SWP^a)*: Top-k Segmental Set Queue and Buffer (TSQB) notation is now activated in the partition window of the slicing panes, where $SWP^a = \{SWP^{a=v} | v \in \{t.a \mid t$ is a tuple in $SWP^a\}\}$ is located. The proposed partition-by-delta attribute $[SW(S)_{\ t\ time} - SW(S, win)_{\ t\ time2} > 0;$ where delta >= query time allocated] is used to specify the timestamps that trigger the sliding process. The equation is defined as $SWP^{a=v} = \{t_i^{a=v} \mid I \in [0 \ldots |SWP^{a=v}|)\}$. The most recent tuple item in the $SWP^{a=v}$ subwindow is represented by $t_0^{a=v}$, while the oldest is by $t_l^{a=v}$. Table 3 demonstrates the progression of the sliding window from $t = 1$ to $t = 10$.

Table 3. The results of the Bucket Instance, BT_k (with sliding window partition)

Timestamp (secs)	Initial Bucket, B_k	SW	Bucket Top-k, BT_k
5	B_2 = {R1}	1	BT_1 = {R1}
10	B_2 = {R2, R3}	1	BT_1 = {R2, R3}
15	B_2 = {R4}	1 & 2	BT_1 = {R4} BT_2 = {R4, R5}
20	B_1 = {R5}; B_2 = {R6};	1 & 2	BT_1 = {R5, R6} BT_2= {R5, R6} *{R1, R2, R3} expired *{R6} pruned
25	B_2 = {R7};	2	BT_2 = {R7}
26	B_1 = {R8}; B_2 = {R9};	2	BT_2 = {R8, R9}
30	B_1 = {R10};	2	BT_2 = {R10}

To identify the top-k potential candidates, the system needs to determine the top-k ranked scores and the aggregated probability distribution among various *UDS* within the segmentation window frame. In this work, we adopt the Delta-based sliding window and with *Definitions 2* and *4*, *SWPTop-kDelta* can be defined as:

*Definition 5 (**Sliding Window Model Top-k Delta, SWMTop-kDelta**)*: Given a *UDS*, a ranking function $\mathcal{F}$ can identify the top-k tuple vectors based on a designated threshold. This threshold generates the score $Prob_{max}(hs_i)$ and probability $Prob_{toplatest}(n)$ across the *SWM* Delta-based mechanism. The scheme continuously monitors the potential candidates $P(s[q]_i) \in W_t$, which have the highest rankings with probability $Prob_{SWMTop\text{-}kDelta}(P(s[q]_i)$, exceeding a combined threshold of $Score_{thresh}$ and $Prob_{thresh}$, in the following manner:

$$Prob_{SWMTop\text{-}kDelta}(P(s[q]_i)) =$$

$$\sum_{p(w)(Wt), P(s[q]i) \in p(w)(Wt)} Prob\{p(w)(Wt)\}.$$

$\mathcal{F}(P(s[q]_i), k, p(w)(Wt)) > Score_{thresh}$ and

$Prob_{thresh}$

The *SWPTop-kDelta* method is an extension of the segmentation window approach used by the *TSBQ* algorithm (as defined in Definition 4). The algorithm lists the top-k potential candidates during the initial phase. For instance, $SWP^a(s_i)$ can be created by P(s[R1-3]), P(s[R4-6]), and P(s[R7-10]). The *SWPTop-kDelta* method uses the top-k vectors and a bucket set that includes specific possible world rules to compute the final top-k result list. Hence, to obtain results affected by the probability distribution $p(w)$, a deterministic *SWMTop-kDelta* query is run on all possible worlds. The segmentation window $SWP^a(s_i)$ achieves the necessary hs before recognizing each tuple item in the sequence. To compute the probability of event *PW* (W1) = {R1, R2, R4, R5} occurring, we need to multiply the individual probabilities of each event. To determine the highest probability of a tuple item, we need to apply the extended generation rules of possible world semantics to each potential candidate within SW. The *SWMTop-kDelta* algorithm computes the probabilities of the top-2 and top-3 tuple items. The process proceeds as follows:

*Definition 6 (**Possible World, p(w)**)*: The possible world rules in *UDS* can be restated as a group of tuples represented by *W*, defined as $\{w_1, w_2, \cdots, w_n\}$. $PW \rightarrow [0,1]$ represents a probability distribution where $\sum_{i=1,\ldots,n} p(w) = 1$, $p(w_i) > 0$. The probability of existence of W is calculated as $p(w_i) = \prod_{i=1}^{n} p(vi) = \prod_{i=1}^{n} \frac{1}{w} = w^{-n}$ The expression $W_w^t(S)$ represents the set of all possible worlds where the sliding window $SW(S, Win)$ is applicable, assuming that all tuples are independent of one another:

$$Prob(W_1) = p_1 * p_2 * p_4 * p_5 = 0.096$$
$$Prob(W_2) = p_1 * p_2 * p_4 * p_6 = 0.024$$

$[p(w) = \prod_{x \in w} p(x) \prod_{x \in w} (1 - p(x)); \quad \sum w \in \Omega \; p(w) = 1]$

In the context of *UDS* that contains semantic possible worlds, there are 2^n possible worlds in the $SW(S)$, where *S* is a set of *n* tuple items. For a positive integer *k* and a possible world $p(w)$, *k* is a set of *k* tuples with the highest scores in *w*, called the top-k tuple of w, denoted as *Hk* (*w*). *T** is the solution to an ongoing top-k query processing on *UDS* is denoted as:

$$T^* = \arg \max_{T} \sum_{\substack{w \in pw \\ Hk(w)=T}} p(w)$$

UDS has two sorting indices that should be considered: the query result's score and probability. The phenomenon is referred to as the transitivity of supremacy and is defined as follows:

*Definition 7 (**Threshold Score Ranking**)***:** When dealing with a top-k query $Q_{p,f}^k$,, where where *P*, *f*, and *k* are all greater than 0, a complete top-k vector contains the highest aggregated probability. This vector's total score is equivalent to $Score_{thresh}$, the designated threshold score. Instead of the standard ranking order, the top-k vector is utilized as an alternative, serving as the select function *Score*. The function *Prob. Score(t)* represents a score ranking function.

Definition 8 (***Threshold Probability Ranking***)**:** The threshold probability, $Prob_{thresh}$, is the sum of probabilities for the top-k vectors with the highest total scores. Let *S* denote the possible world space Ω, where *k* is a positive integer. *Prob(t)* is a probability function, and Top-k(*W*) is a collection of *k* tuple items generated from the possible world *W* based on the scoring function *Score*(*t*). It can proceed by comparing tuple item t_i to t_j, when $Score(t_i) > Score(t_j)$ and $Pro(t_i) > Pro(t_j)$, denoted by $t_i \succ t_j$.

Concerning the threshold assumptions and definitions mentioned above, the k-vectors with the most significant values can be defined in a particular way:

Definition 9 (***Discovering the k-vectors with the highest values***): When dealing with an uncertain top-k query, one can consider a top-k vector rule (v_i). First, a set of k-tuple items is selected based on their ranking according to the notations of $q(hs_i)$, $Prob_{max}(hs_i)$, and $Prob_{toplatest}(n)$. The maximum values of these items are described as follows:

1. For each iteration *i*, the criteria for hs_i belonging to *X* are considered. The expression $q(hs_i)$ represents the chance that a *v*-tuple of $q(hs_i)$ is absent. This can be summarized as $[q(hs_i) = 1 - \sum_{tj \in hsi}{}^{t} p(t_j)]$, where $p(t_j)$ is the probability of the original tuple item t_j occurring in the possible world.
2. The original tuple item in set hs_i with the highest probability is represented as $Prob_{max}(hs_i)$. The aggregated top-k vector with the highest

probability (which becomes a full top-k vector) is denoted as $Prob_{toplatest}(n)$.

Definition 10 (***Optimization using Clearing***, ***OC***): The task entails effectively executing the *OC probclear*(τ_l) operation on a *v*-tuple τ_l, with an OC-probability denoted as *Prob*(*l*) ($0 \leq Prob(l) \leq 1$). The purpose is to ensure that *OC probclear*(τ_l) can achieve a successful performance with a probability of *Prob*(*l*). If *probclear*(τ_l) is successful, τ_l is substituted with a *v*-tuple that consists of a single tuple *τl*. If the attempt to clear the probability (*τl*) is unsuccessful, *τl* will stay unchanged.

4.2 Problem Statement

Processing Top-k queries over uncertain data streams has emerged as a promising approach to developing an intuitive information system. Data's underlying and unavoidable uncertainty is typically attributed to unexpected and unreliable signal readings, sensor update delays, or insufficient expertise. Uncertainty data mainly refers to factors such as indeterminacy, unreliability, unpredictability, randomness, inconsistency, variability, incompleteness, unknown bounding, and irregularity. Our current research focuses on the challenges and uncertainties surrounding top-k query processing in Real-Time Traffic Management applications. These issues are particularly prominent and require careful examination.

Following the above research problem, we have identified three challenges that need to be addressed in this thesis, focusing on analyzing and computing Top-k on uncertain data streams. **Limitation 1**: An efficient implementation of a sliding window approach is needed to handle Top-k queries on uncertain data streams. If the sliding window model is not employed correctly, candidate tuple items within the window frame may cause significant overlapping computing costs. Therefore, it is essential to ensure that the sliding window approach is used effectively. **Limitation 2**: An efficient algorithm is needed to improve the retrieval of the top-k query results from uncertain data streams, particularly when computing the top-k score and probabilities expected to have significant computational expenses for generating the set of possible worlds. **Limitation 3**: The number of possible world instances grows exponentially, affecting top-k continuous query processing time to be high.

5 Proposed Top-k Query Processing Framework

To achieve the primary goal, we offer a framework called the SWMTop-kDelta. This framework has three stages: In the first stage, we will use the necessary SWM representation to convert a set of tuple elements into a window fragment. Phase II is essential in determining the optimal processing time utilizing the suggested algorithm. This aspect focuses on the complexity of top-k computation, which is directly influenced by factors such as continuous queries, score value, probability, timestamp interval (delta-attribute), and possible world rules. Phase III builds upon the information obtained in Phases I and II. It implements optimization techniques to prevent unnecessary top-k computations when determining and computing possible world vector rules.

5.1 Phase I

It addresses the challenge by introducing a condition, expressed as [$SW(S)_{t\ time} - SW(S,win)_{t\ time2} > 0$; where *delta* >= query time]. The framework design of Phase I is depicted in Figure 1. The study presents the development of a method for enhancement by utilizing the *delta* attribute with a constant timestamp as a slide value.

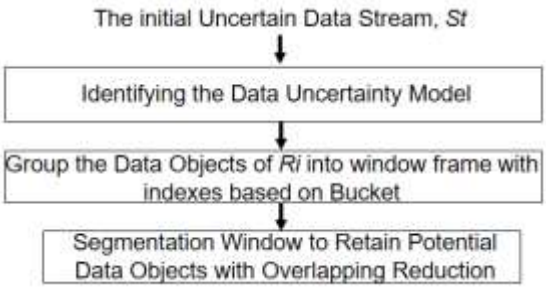


Fig. 1: SWMTop-kDelta Flow for Utilizing SWM Over the Initial UDS – Phase I

As illustrated in Table 4 and Figure 2, the computations on registers R6, R5, R4, R3, R2, and R1 are entered into the sliding window frame based on the Delta-based sliding condition mentioned above, with *SW*=1. The computation for the expression *SW*=2 will be performed on registers R10, R9, R8, R7, R5, and R4. However, registers R3, R2, and R1 will be excluded as a later tuple item will push them. The candidate R6 was eliminated due to its lowest score among all candidates, making it ineligible for the top spot.

Table 4. Example of tuple items from UDS

Timestamp (secs)	**5**	**10**	**10**	**15**	**20**	**20**	**25**	**26**	**26**	**30**
Tuple items	R1	R2	R3	R4	R5	R5	R7	R8	R9	R10
score	80	65	45	30	50	50	40	55	78	90
prob	0.3	0.4	0.5	1	0.8	0.8	0.5	0.6	0.4	0.8
SW	1	1	1	1|2	1|2	1|2	2	2	2	2
index	1	2	3	4	5	5	7	8	9	10

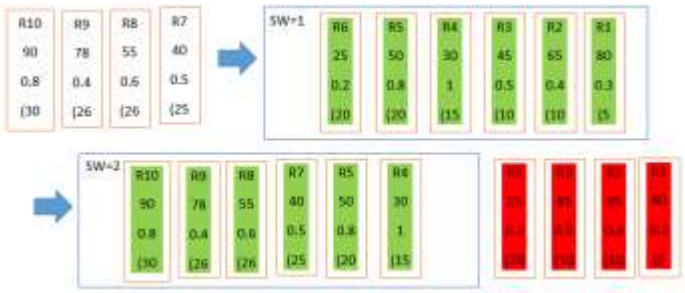


Fig. 2: *SWM* (Delta-based - tuple insertion & exit)

The algorithm described in Figure 3 is based on *or*-set probability distribution, scoring, and tuple-level confidence values. In the process, each bucket B_k will store the following information and a timestamp: (i) The bucket number with a probability level greater than 5.0 will be retained on the bucket list; (ii) The bucket number with the highest score but a probability level less than 5.0 will also be kept on the bucket list; (iii) If a bucket contains items expiring quickly, it will be removed if the sliding window period is exceeded. This is because it falls below the confidence level boundaries and has a low-rank score.

The GML section serves two primary purposes. Firstly, it presents the grouping of tuple items with indexes, I_j ($1 \leq j \leq s$), over *UDS*. Secondly, the list of attributes of the *GML* synopsis should include all tuple items $S[q]$ from *UDS* being considered for the Top-k candidate list before the period expires. In the selection process, B_k is used to efficiently retrieve all tuple items (x) from the repository Bucket Top-k, BT_k SW and assign their corresponding values to each segmentation group. This feature enables the efficient computation of *UDS* attributes, where $t_i \in SW$ [t-|*SW*|+1, t]. To maintain data tracking for each k, an index structure called I_j will be created over *SW* for characteristics. This will enable us to scan the group bucket B_k only once. The solution presented here eliminates the expense of refuse collection caused by empty buckets, as BT_k is always guaranteed to possess a value. Figure 3 depicts the procedure described. This scanning process results in the creation of two sets, namely *I* BT_1 {R1 → d_j , R2 → d_j, ..., R6 → d_j} and *I* BT_2 {R4→ d_j , R7 → d_j, ..., R10 → d_j}. It is important to note that the access order is insignificant in this context.

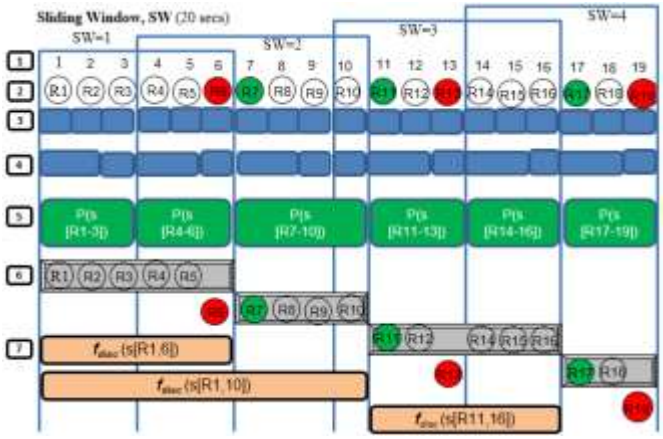
Input: Bucket of Instance, $B = \{B_1, B_2, \cdots, B_i \ldots\}$
Output: Bucket Top-k, $BT_k = \{BT_1, BT_2, \cdots, BT_i \ldots\}$
1. **Begin**
2. $t = 0$; Initial parameters
3. **While** *UDS* is active **do**
4. t++;
5. Read a new data tuple (x, *Bi*), map to index, *d*;
6. **If (***d* is mapping to membership density group in B_i) then
Create *SW* of *d,* and insert into BT_k*;*
7. **Else Update**
BT_k according to check expired/removed *x*
SW from membership group in B_i
8. **End While**
9. **End**

Fig. 3: The Algorithm to Output Bucket of Top-k, BT_k

To reduce the processing complexity of developing rules for all possible worlds to almost a minimum, it is necessary to further segment the sliding window between tuple items from each bucket. The Top-k Segmental Set Queue and Buffer (*TSQB*) is constructed over BT_k tuple items. Since all bucket sets are used by the *SWMTop-kDelta* method, only the oldest bucket set, BT_k, is required to retrieve the top-k query results. When many sliding windows overlap, it increases computational complexity and takes longer to process data.

Theorem 1: When the *TSQB* algorithm and the preceding algorithms are executed, each tuple item $P(s[q]) \in UDS$ will be identified as one of the Top-*k* potential candidates. This will occur every time the sliding window is moved in each composition, which is determined by the discretization function.

Proof 1: To minimize the processing complexity associated with creating rules for all possible worlds, it is essential to further divide the sliding window between tuple items within each bucket. Our proposed *SWMTop-kDelta* approach utilizes all bucket sets, but only the oldest bucket set, BT_k, is necessary for retrieving the top-k query results. The Top-k Segmental Set Queue and Buffer (*TSQB*) method is built on BT_k tuple components, as seen in Figure 4 and Figure 5.



1. Indexing 2. Tuple Items 3. Panes Breakdown, *p1* 4. Segmentation, *p2* (*p1* = Sliding – *p2*)	5. Periodic Top-k (potential candidates) 6. Segmental Buffer 7. Discretization Function, f_{disc}

Fig. 4: An Overview of TSQB

Input: Bucket Top-k, $BT_k = \{BT_1, BT_2, \cdots, BT_i \ldots\}$
Output: Top-k Segmental Sets Queue, Top-k Ψ and Buffer, $SB(S_{l1}) = \{SB(S_{l1}), \ldots, SB(S_{ln}) \ldots\}$
1. **Begin** Tuple set $s_i = \emptyset$; segmental set queue, Ψ = ∅; let *B* be a buffer with size *kH*;
2. *t* = 0; Initial parameters from s_i
3. **For** (each arriving tuple *t* from BT_k)
4. Group-by tuple *t* with *tid*; //partition with summation *tid* is generated;
5. insert *t* into s_i;
6. **Begin**
7. **If** (successfully create a segmental sets $SWP^a(s_i)$ for s_i)
8. append $SWP^a(s_i)$ to Ψ;
9. remove tuples in s_i older than *t*″ (including *t*″), where *t*″ is the oldest tuple in $SWP^a(s_i)$;
10. **For** (each segmental sets $SWP^a(s_i) \in \Psi$ from new to old)
11. **Execute Final *SWMTop-kDelta* Algorithm**;
12. **If** (*t* become highest ranked tuple in $SWP^a(s_i)$)
13. **Update** $SWP^a(s_i)$
14. insert t into *SB*;
15. **If** (*SB* is full);
16. find the smallest *i* such that SB_i // Admits a segmental set;
17. starting from *i*, build SWP^a on *SB*;
18. update the existing SWP^a;
19. *SB* = ∅;
20. **If** ($SWP^a(s_i)$ is affected)
21. update $SWP^a(s_i)$;
22. **Else**
23. **break**;
24. **If** (the expiring tuple ∈ $SWP^a(s_i)$)
25. remove $SWP^a(s_i)$ from Ψ; //remove Expired segmental sets queue in
26. $SWP^a(s_i)$:= first segmental sets queue in Ψ;
27. compute the array r on the new $SWP^a(s_i)$
28. **End**
29. **End**

Fig. 5: The TSQB algorithm

The acquisition and utilization of *TSQB* properties are necessary for the subsequent stages of the *SWMTop-kDelta* algorithm's execution. The Top-k potential candidates are obtained from the BT_k bucket by performing the following operations:

1. **Property 1** (Breakdown): The number of subwindows in $|SWP^a|$ is represented by SWP^a and a sub-window linked to the value of the partition-by attribute *v* is expressed as $SWP^{a=v}$.
2. **Property 2** (Hybrid Method): The technique of segmental slicing is an orthogonal approach that can be employed with the buffer set to enable an implicit class of windows, which is a superclass of periodic windows, while still maintaining the continuous appearance of the *SWM*.
3. **Property 3** (Buffering): The first component, contains the current top-k potential candidates set, which satisfies the condition $Pr_{SWMTop\text{-}kDelta}(S[q]) > \alpha$); Secondly, data buffering synopsis built on SWP^a, where $SB(S_{ln})$ represents the set of tuple items that are not currently top-k potential candidates where $0 < Pr_{SWMTop\text{-}kDelta}(S[q]) \leq \alpha$ holds at delta attribute).

Table 5 clearly shows the total *TSBQ* results for segmentation and merging multiple overlapping *SW*. An example of this is the $SWPa(s_i)$ which is established by *P*(*s*[R1-3]), *P*(*s*[R4-6]), *P*(*s*[R7-10]), *P*(*s*[R11-13]), *P*(*s*[R14-16]), and *P*(*s*[R17-19]). During a basic evaluation segmentation, it is observed that *P*(*s*[R4-6]) includes BT_1 = {R4, R5, R6} and BT_2 = {R4, R5, R6}. As a result, it is necessary to invoke this function twice, once for *SW*=1 and once for *SW*=2. Increased window overlap results in a decrease in the number of operations required for each window merging process.

Table 5. The results of the $SWP^a(s_i)$

Potential Candidates	*SW*	Bucket Top-k, BT_k	Segmental Buffer, SB_k
P(s[R1-3])	1	BT_1 = {R1, R2, R3}	SB_1 = {R1, R2, R3, R4, R5, R6} *{R6} pruned
P(s[R4-6])	1&2	BT_1 = {R4, R5, R6} BT_2 = {R4, R5, R6}	
P(s[R7-10])	2&3	BT_2 = {R7, R8, R9, R10}	SB_2 = {R7, R8, R9, R10}
P(s[R11-13])	3	BT_3 = {R11, R12, R13}	SB_3 = {R11, R12, R13, R14, R15, R16} *{R13} pruned
P(s[R14-16])	3&4	BT_3 = {R14, R15, R16} BT_4 = {R14, R15, R16}	
P(s[R17-19])	4	BT_4 = {R17, R18, R19}	SB_4 = {R17, R18, R19} *{R19} pruned

5.2 Phase II

Phase II will first clarify the explanation and actions in obtaining the top-k potential candidates. This phase proposed three key elements, and each case was specifically designed to address a different score and probability scenario in *SWM*. The proposed framework can be implemented by following the procedures depicted in Figure 6. The utilization of *Tuple-level Uncertainty* (*TLU*) to generate possible world semantics, reveals that the *UDS* encompasses

a total of 12 possible worlds (illustrated in Table 6 based from Table 2). The feasibility of providing answers to top-*k* queries relies on probability-based considerations.

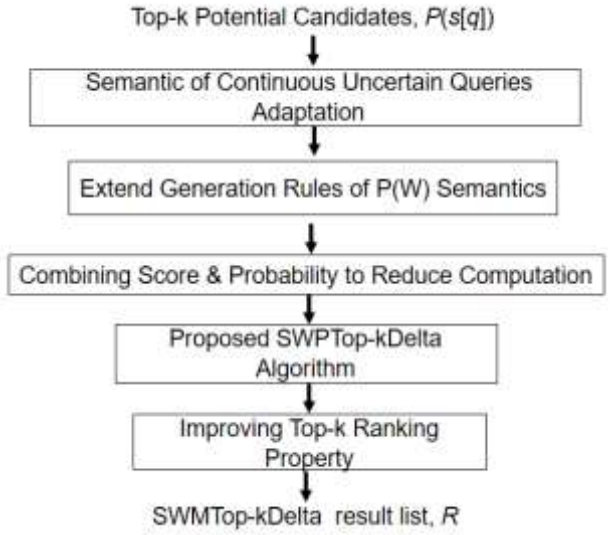


Fig. 6: The Proposed SWMTop-kDelta Framework for Utilizing SWM Over the Initial UDS – Phase II

Table 6. Evaluation of query Q: $\sigma_{score \geq 40}$ (UDS) and within initial 20 seconds, which returns tuple items whose score (speed) is >= 40

possible world *p(w)*	probability of *W*	possible answer	answer probability
$\{t_1, t_2, t_4, t_5\}$	$p_1 * p_2 * p_4 * p_5 = 0.096$	$A_1 = \{t_1, t_2, t_5\}$	$P_1 = 0.096+0.024 = 0.12$
$\{t_1, t_2, t_4, t_6\}$	$p_1 * p_2 * p_4 * p_6 = 0.024$		
$\{t_1, t_3, t_4, t_5\}$	$p_1 * p_3 * p_4 * p_5 = 0.12$	$A_2 = \{t_1, t_3, t_5\}$	$P_2 = 0.12+0.03 = 0.15$
$\{t_1, t_3, t_4, t_6\}$	$p_1 * p_3 * p_4 * p_6 = 0.03$		
$\{t_1, t_4, t_5\}$	$p_1 * (1-p_2-p_3) * p_4 * p_5 = 0.024$	$A_3 = \{t_1, t_5\}$	$P_3 = 0.024+0.006 = 0.03$
$\{t_1, t_4, t_6\}$	$p_1 * (1-p_2-p_3) * p_4 * p_6 = 0.006$		
$\{t_2, t_4, t_5\}$	$(1-p_1) * (p_2) * p_4 * p_5 = 0.224$	$A_4 = \{t_2, t_5\}$	$P_4 = 0.224+0.056 =0.28$
$\{t_2, t_4, t_6\}$	$(1-p_1) * (p_2) * p_4 * p_6 = 0.056$		
$\{t_3, t_4, t_5\}$	$(1-p_1) * (p_3) * p_4 * p_5 = 0.28$	$A_5 = \{t_3, t_5\}$	$P_5 = 0.28+0.07 = 0.35$
$\{t_3, t_4, t_6\}$	$(1-p_1) * (p_3) * p_4 * p_6 = 0.07$		
$\{t_4, t_5\}$	$(1-p_1) * (1-p_2-p_3) * p_4 * p_5 = 0.056$	$A_6 = \{t_5\}$	$P_6 = 0.056+0.014$ 0.07
$\{t_4, t_6\}$	$(1-p_1) * (1-p_2-p_3) * p_4 * p_6 = 0.014$		

Theorem 2: If the stopping condition $Prob_{toplatest}(n)$ is met, it is possible to identify the top-k vector with the highest combined probability of being the top-k answer for a v-relation of X.

$$Prob_{toplatest}(n) \geq \prod_{1 \leq i \leq n} max$$

$$\{ Prob_{max}(hs_i), q(hs_i) \}$$

Proof 2: To obtain the top-k result list with the highest aggregated probability full vector (*FV*), it is necessary to realign the top-k potential candidates for generating the *k*-ReduceSet for *UDS*. This is accomplished by using a data structure known as *ReducePSW* where it captures and records the $q(hs_i)$ tuple items for each member of the segmental sets *SWPa*(*si*) in *UDS*. The *k*-ReduceSet can be obtained by scanning the *UDS* once within a sliding window segment. The time and space complexity of the process is $O(|UDS|^2)$. Using examples from Table 7 and Table 8, the summary of the *k*-ReduceSet can be explained as follows:

i. *P*(s[R1-3]): 1-ReduceSet = $\{t_1\}$; 2-ReduceSet = $\{t_1, t_2\}$; *k*- ReduceSet = $\{t_1, t_2, t_3\}$ where $k \geq 3$;
ii. *P*(s[R4-6]): 1-ReduceSet = $\{t_5\}$; 2-ReduceSet = $\{t_4, t_5\}$; *k*- ReduceSet = $\{t_4, t_5, t_6\}$; where $k \geq 3$.

Table 7. The New Vector of Combination Tuple Items in v_i

Potential Candidates	Bucket Top-k, BT_k	Pair Combination of (*Score*(v_i), *Prob*(v_i)) Tuples in v_i
P(s[R1-3])	$BT_1 = \{t_1, t_2, t_3\}$	v_1 = (80, 0.3); v_2 = (65, 0.28); v_3 = (45, 0.07);
P(s[R4-6])	$BT_2 = \{t_4, t_5, t_6\}$	v_4 = (30, 0.014); v_5 = (50, 0.336); v_6 = (25, 0);

Table 8. The process of generating k-ReduceSet

Position	Initialize	Retrieve $\{t_1\}$	Retrieve $\{t_2\}$	Retrieve $\{t_3\}$
ReducePSW [t_1]	Ø	Ø	Ø	Ø
ReducePSW [t_2]	Ø	Ø	$\{t_1\}$	$\{t_1\}$
ReducePSW [t_3]	Ø	Ø	Ø	$\{t_1, t_2\}$
Position	**Initialize**	**Retrieve $\{t_4\}$**	**Retrieve $\{t_5\}$**	**Retrieve $\{t_6\}$**
ReducePSW [t_4]	Ø	$\{t_5\}$	Ø	$\{t_5\}$
ReducePSW [t_5]	Ø	Ø	Ø	Ø
ReducePSW [t_6]	Ø	Ø	Ø	$\{t_4, t_5\}$

Theorem 3: The top-*k* result list comprises tuple items with the highest ranked score and aggregated probability combination of two top-k vectors, v_i and v_j. These combinations are represented by (*Score*(v_i), *Prob*(v_i)) and (*Score*(v_j), *Prob*(v_j)), respectively. Assuming that v_i outperforms v_j, denoted as $v_i > v_j$, the algorithm ensures the logical chain of conditions as described below:

i. *Score*(v_i) > *Score*(v_j) where *Prob*(v_i) == *Prob*(v_j)
ii. *Score*(v_i) == *Score*(v_j) where *Prob*(v_i) > *Prob*(v_j)
iii. *Score*(v_i) > *Score*(v_j) where *Prob*(v_i) > *Prob*(v_j)

Proof 3: This algorithm is represented by {(*Score*(v_1), *Prob*(v_1), …, (*Score*(v_n), *Prob*(v_n))}, which combines the top-k ranked scores and aggregated probabilities for a set of n likely top-k vector rules {vi, ..., vn}. Therefore, to establish the supremacy of v_i as the top-*k* vector rule *p*(*w*), it is essential to demonstrate that there is no other vector $v_j \in \{v_1, ..., v_n\}$, where $v_j \neq v_i$, that surpasses v_i in supremacy. The proposed *SWMTop-kDelta* algorithm approach is depicted in Figure 7. This approach builds upon the prior description in *Phase* I and demonstrates superior performance compared to the *Naive* algorithm method.

The operation commences by executing the proposed window segmentation for every incoming tuple item *t* from BT_k as the sliding window progresses with the initial state of *v*-tuple items *T*. If *Prob*(*hs*) < 1, the *v*-tuple is unlikely to occur, and none of the elements in *hs* will perform. To compute possible permutations of world rules for the variables *Score*(*v''*) and *Prob*(*v''*), you need to add the list of array *hs* to *v*. This completes the cycle of generating new vector rules for *v''*. Moreover, *Prob*(v) X (1 - *Prob*(*hs*)), where (1 - *Prob*(*hs*)), where (1 - *Prob*(*hs*)) represents an ideal set of virtual tuple items possessed by *hs*.

Input: v-tuples T (Onsite tuples of Table 8)
Output: Top-k vectors rules of p(w) has the supreme ranked score
1. **Begin** Tuple set of *Score*(v_0) = 0; *Prob*(v_0) = 1;
2. NFV = {v_0} //not full vector set (length < k);
3. NFV′ = {∅};
4. FV = {∅} // full vector set (length = *k*);
5. Computation Theorem 2 // means *hs* performs
6. **While** NFV ≠ {∅}
7. **For** (each arriving v-tuple *hs* from BT_k) **do**
8. **For** each tuple *t* in v-tuple *hs* **do**
9. **For** each not full vector *v* in NFV **do**
10. Append *t* to *v* to get a new vector *v′*
11. Compute *Score*(*v′*) and *Prob*(*v′*) for Top-2 and Top-3
12. *Score*(*v′*) = *Score*(*v*) + *Score*(*t*)
13. *Prob*(*v′*) = *Prob*(*v*) + *Prob*(*t*)
14. Computation Theorem 3 // Make sure that *v′* is a redundant vector
15. **If** (*v′*) is not being redundant
16. **If** (*v′*) is not full vector rules then
17. *Add v′* to NFV'
18. **Else**
19. *Add v′* to FV
20. **End if**
21. **Else**
22. Computation *ReducePSW* (*v′*)
23. Computation Pruning Strategies (*v′* from ϒ(*t*))
24. **End if**
25. **End for**
26. **End for**
27. **If** *Prob*(*hs*) < 1
28. Append *hs* to *v* to acquire new vector rules of *v''* // means that none of *t* ∈ *hs* performs
29. Compute *Score*(*v''*) and *Prob*(*v''*) for Top-2 and Top-3
30. *Score*(*v''*) = *Score*(*v*) + *Score*(*t*)
31. *Prob*(*v''*) = *Prob*(*v*) x (1 - *Prob*(*hs*))
32. Computation Theorem 3 // Make sure that *v'* is a redundant vector
33. **If** (*v''*) is not a redundant vector
34. **If** (*v'*) is not full vector rules then
35. *Add v''* to NFV'
36. **Else**
37. *Add v''* to FV
38. **End if**
39. **Else**
40. Computation *ReducePSW* (*v′*)
41. Computation Pruning Strategies (*v′* from ϒ(*t*))
42. **End if**
43. **End if**
44. NFV = NFV′
45. NFV′ = {∅}
46. **End for**
47. **End While**
48. Computation Theorem 4 // Where supreme ranked score of each $v_j \in FV$ will return vector rules of having the highest ranked score

Fig. 7: The main algorithm of the framework for SWMTop-kDelta query answering – Phase II

Upon arrival, the probability method determines whether an object should be included in the top-k candidate objects for a given query, as indicated in Table 9. The segmentation window $SWP^a(s_i)$ achieves the necessary *hs* before recognizing each tuple item in the sequence. To compute the probability of event *PW* (W1) = {R1, R2, R4, R5} occurring, we need to multiply the individual probabilities of each event. The probabilities for R5 and R6 must sum up to 1, and the total number of possible worlds can be calculated by multiplying the values of each factor: 2 x 3 x 1 x 2 = 12.

To determine the highest probability of a tuple item, we need to apply the extended generation rules of possible world semantics to each potential candidate within sliding windows. The sum of the probabilities of R5 occurring among the top-3 can be obtained by summing 0.12+0.024+0.224+0.28+ 0.056=0.704, as illustrated in Table 10. The query, which includes a top-2 clause with a condition where *SW*=1, yields R5 and R2 as the current results from $SWP^a(s_1)$ and $SWP^a(s_2)$ occurrences. The *SWMTop-kDelta* algorithm computes the probabilities of the *top*-2 and *top*-3 tuple items. The *top*-2 probability for R5 represents the number of possible worlds in which R5 is ranked within the top 3 among the possibilities of W3, W5, W7, W9, and W11. The cumulative probabilities of R5 and R2 are presented in the third column of Table 11, which is indicated by bold font.

Table 9. The Computation Probability of PW (SW=1) using v-Tuple, T Onsite approach

PW	Calculation Prob.	Prob.	Top-2	Top-3
W1= R1, R2, R4, R5	0.3 X 0.4 X 1 X 0.8	0.096	R1, R2	R1, R2, R5
W2= R1, R2, R4, R6	0.3 X 0.4 X 1 X 0.2	0.024	R1, R2	R1, R2, R4
W3= R1, R3, R4, R5	0.3 X 0.5 X 1 X 0.8	0.12	R1, R5	R1, R5, R3
W4= R1, R3, R4, R6	0.3 X 0.5 X 1 X 0.2	0.03	R1, R3	R1, R3, R4
W5= R1, R4, R5	0.3 X (1-0.4-0.5) X 1 X 0.8	0.024	R1, R5	R1, R4, R5
W6= R1, R4, R6	0.3 X (1-0.4-0.5) X 1 X 0.2	0.006	R1, R4	R1, R4, R6
W7= R2, R4, R5	(1-0.3) X 0.4 X 1 X 0.8	0.224	R2, R5	R2, R4, R5
W8= R2, R4, R6	(1-0.3) X 0.4 X 1 X 0.2	0.056	R2, R4	R2, R4, R6
W9= R3, R4, R5	(1-0.3) X 0.5 X 1 X 0.8	0.28	R5, R3	R5, R3, R4
W10= R3, R4, R6	(1-0.3) X 0.5 X 1 X 0.2	0.07	R3, R4	R3, R4, R6
W11= R4, R5	(1-0.3) X (1-0.4-0.5) X 1 X 0.8	0.056	R5, R4	R5, R4
W12= R4, R6	(1-0.3) X (1-0.4-0.5) X 1 X 0.2	0.014	R4, R6	R4, R6

Table 10. SW=1 consist tuple item for Top-2 and Top-3 probability

ID	Top-2 Prob. Calculation	Top-2 Prob	Top-3 Prob. Calculation	Top-3 Prob
R1	0.096+0.024+0.12 +0.03+0.024+0.00 6	0.3	0.096+0.024+0.12 +0.03+0.024+0.00 6	0.3
R2	0.096+0.024+0.22 4+0.056	0.4	0.096+0.024+0.22 4+0.056+	0.4
R3	0.03+0.28+0.07	0.38	0.12+0.03+0.28+0 .07	0.5
R4	0.006+0.056+0.07 +0.056+0.014	0.202	0.024+0.03+0.024 +0.006+0.224+0.0 56+0.28+0.07+0.0 56+0.014	0.784
R5	0.12+0.024+0.224 +0.28+0.056	0.704	0.096+0.12+0.024 +0.224+0.28+0.05 6	0.8
R6	0.014	0.014	0.006+0.056+0.07 +0.014	0.173

Table 11. Top-2 and Top-3 Result List based on Probability

Top-2 Prob	Top-3 Prob
R5	**R5**
R2	**R4**
	R3

5.3 Phase III

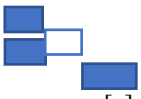
The focus of this phase is to propose SWM operations to enhance the efficiency of top-k query processing. The framework comprises two key elements:

i. Clearing the possible world;

ii. Implementing pruning strategies on an SWM and top-k potential candidates to compute sort-rank tuple items.

Thus, implementing *PWC* as an additional mechanism on a large scale can effectively reduce the time complexity from exponential to polynomial. Specifically, this reduction is achieved by considering the size of each segmental *UDS*. Two cases need to be considered as deliberated in Table 12 and Figure 8.

Table 12. The two Cases of an probclear(τ_l) optimization with explanation

***UDS* state**	**Before *probclear*(τ_l)**	**After *probclear*(τ_l)**	**Explanation**
Case 1 z_1 z_2 z_3 (9 possible world vector rules for τ_1, τ_2, τ_3)	 $\tau_1, \tau_2, ..., \tau[x$	 $\tau_1, \tau_2, ..., [x]$	*probclear*(τ_l) is performed for U_l times, the uncertainty of τ_l with the number of times performed for M_l
	X = { τ_1, τ_2}, so it can be $Z_1 = (\tau_1, \{t_1\}, \{t_2\})$; $z_2 = (\tau_2, \{t_3\}, \{t_4\}, \{t_5\}, \{t_6\})$; and $z_3 = (\tau_3, \{t_7\})$. Let $\rightarrow X \in z_1 \times z_2$.	If $t_i \in \tau_1$ and *probclear*(τ_l) is successful, the probability Let $\rightarrow X(l) = \{t_i\}$ is similar to $e_i{}^n$. Result: $(\tau_1, \{t_3\})$, $(\{t_1\}, \{t_4\}, \{t_6\})$, and (τ_1, τ_2). $\{t_5\}$ is cleared from τ_2.	To derive the optimization solution for the clearing problem, it is necessary to determine the best possible world vector rules for *probclear*(τ_l).
Case 2: 9 possible world vector rules for τ_1, τ_2, τ_3	τ_1 to be cleared equals to $\sum_{ti \in \tau l} pi / k$ where top-k probability of each v-tuple. Therefore, the v-tuple is more likely to be cleaned.	Let $cost_l$ represent the computational effort required to execute the *probclear*(τ_l) function once. Let "O" denote the allocated memory, specifically for removing unnecessary vector rules.	Operation is based on the anticipated optimization query processing improvement level and the associated cost.

The achievement of this task involves the execution of *probclear*(τ_l) between the output of the

clearing algorithm, depicted in Figure 9, which is implemented based on the two cases that were previously deliberated.

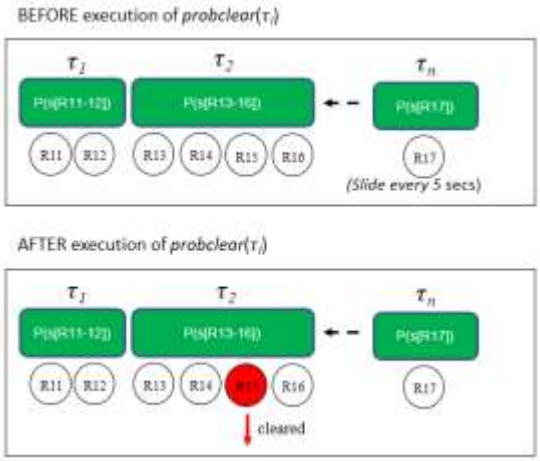


Fig. 8: Clearing operation based on Case 1 is performed successfully

Input: Top-k Segmental Sets Queue, Top-k^m Ψ
Output: $Z^n = \{ \tau_1, \tau_2, ..., \tau[X] \}$
1. **Begin**
2. **For** (each data tuple X consisting $t_i \in \tau_1$ from Top-k Ψ)
3. Let $LZ_l = (\tau_1, \{t_1\}, \{t_2\}, \ldots, \{t_n\})$, the clearing data tuples of t_i $\in \tau_1$
4. **For** (each data tuple X consisting $t_i \in \tau_1$)
5. **Begin**
6. **If** (successfully create τ_1 sets *of probclear*(τ_l) for t_i)
7. Remove < LZ_l with low probability> tuples with low probability in LZ_l but remain single tuple item, U_l times
8. Append t_i to v to get a new vector v' (means *probclear*(τ_l)
9. **Else**
10 **If** ($\tau_1 >= Z_{costl} > pi / k$) **then** only when O has high memory
11. Insert tuples with high probability in Z and remain array tuple of t_i that belong to Top-k Ψ
12. Append t_i to v to get any remaining vector v' within adjacent window
13. **End**
14. **End**
14. Bucket of Z^n = *probclear*($\{ \tau_1, \tau_2, ..., \tau[X]\}$)
15. **End**

Fig. 9: The Clearing Algorithm with PWC Indicator

Table 13 shows a sample of the improved Bucket of Z^n. The bucket consists of a sliding window that moves every 5 seconds. Each movement of the window demonstrates how the Clearing algorithm transforms tuple items, resulting in the tuples (τ_1, $\{t_3\}$), ($\{t_1\}$, $\{t_4\}$, $\{t_6\}$), and (τ_1, τ_2). $\{t_5\}$ is cleared from τ_2.

Table 13. Function probclear(τ_l) of assigned a higher rank if its score and probability surpass 0.4

id	C Task (slide for 5 secs)	Cleared (yes|no)	Marginal Prob.
t_1	If 35 secs, t_1 score > t_2 and prob >= 0.4	no	y(0.5) n(0.5)
t_2	If 35 secs, t_2 score > t_1 and prob >= 0.4	no	y(0.4) n(0.6)
t_3	If 40 secs, t_3 score > t_4 and prob >= 0.4	no	y(0.5) n(0.5)
t_4	If 40 secs, t_4 score > t_3 and prob >= 0.4	no	y(0.5) n(0.5)
t_5	If 44 secs, t_5 score > t_6 and prob >= 0.4	yes	0
t_6	If 44 secs, t_6 score > t_5 and prob >= 0.4	no	y(0.7) n(0.3)
t_7	If 50 secs, t_7 score > t_{n+1} and prob >= 0.4	no	y(1) n(0)

The concept of pruning tuple items within a sliding window can be explained through different methods of identifying the top-k (top-2 and top-3) potential candidates. In this example, the elements indexed as 1-2 depend on their $SWP^a(s_i)$, and each index is associated with two segmentation frames. When evaluating a sliding window that moves every 5 seconds, the earliest element (R1) will be removed when the succeeding elements (R7, R8, R9, and R10) enter. However, when SW=1, the window will still have the other elements. Let's assume that {R6} is one of the elements in the sliding window currently being evaluated for pruning. To determine whether at least two elements within the window are more recent and greater than {R6}, we need to compare {R6} with each window component. According to the given condition, it is clear that {R6} will not be selected as one of the top two options in the current scenario where SW=1.

To perform sliding window frame pruning, it is essential to establish and clarify Theorems 5 and 6:
Theorem 4: Figure 10 illustrates if there are at least *RankF* elements in the sequence of an array {$e(m+1)$, $e(m+2)$, ..., $e(N)$} that are all greater than $e(m)$, then $e(m)$ is not eligible to be included in the *RankF* set for future incoming tuple items of the current $SWP^a(s_i)$ window frame. However, it is still possible to incorporate it within the complete sliding window frame, which represents the total duration before the window recommences sliding.

Proof 4: To satisfy $e(m)$ within the sliding window, it needs to ensure that there are at least *RankF* elements in the sequence {$e(m+1)$, $e(m+2)$, ..., $e(N)$} that are greater than $e(m)$. Additionally, we

require the sequence {$e(m+1)$, $e(m+2)$, ..., $e(N)$} and $RankF$ + 1. Assuming that the level $e(m)$ appears before any other element in the sequence, except for a certain number of (m-1) arbitrary elements, we can determine this by comparing the timestamp attributes for the exit policy. In this scenario, the iterations of the sliding window that contain $e(m)$ will include the sequence $e(m+1)$, $e(m+2)$,..., $e(N)$} in the order they appear. It's important to note that the number of iterations cannot be fewer than the sequence {$e(m+1)$, $e(m+2)$,..., $e(N)$} in the order of time. Therefore, in subsequent iterations of the sliding window frame, the element $e(m)$ will consistently be excluded as a $RankF$ element.

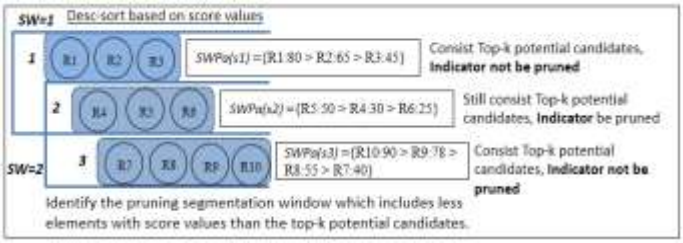


Fig. 10: The Initial sliding window frame with elements before $SWP^a(s_i)$ pruning

Theorem 5: Figure 11 illustrates for each subsequent instance of the $SWP^a(s_i)$, element $e(m)$ should not possess a $RankF$ if there are at least (N − $RankF$ +1) elements in the sequence $e(m+1)$, $e(m+2)$,..., $e(N)$ that are smaller than or equal to $e(m)$ but have the possibility of being among the top k candidates in the entire sliding window frame, which represents the time required before the window resumes sliding.

Proof 5: To determine the variable timestamp for elements on $RankF$ in the sliding window, we can use the sliding window semantics of delta (attribute, delta). $RankF$ of $e(m)$ refers to the position of $e(m)$ in the sliding window that contains $e(m)$, $e(m+1)$, $e(m+2)$,..., e(N) and other (m−1) elements can be limited to a range of 1 to (N − (N − $RankF$ +1)) = $RankF$ − 1. This is because there is a minimum of (N − $RankF$ +1) elements in the sequence $e(m+1)$, $e(m+2)$,..., $e(N)$ that are less than or equal to $e(m)$. In any subsequent occurrence of the sliding window containing $e(m)$, the sequence $e(m+1)$, $e(m+2)$,..., $e(N)$ will also be present. This is because $e(m)$ comes before all the elements in the series (m+1), $e(m+2)$,..., $e(N)$. Therefore, $e(m)$ is not a part of $RankF$ and cannot be included.

Again, this study suggests an efficient optimization method to process only the latest tuple, prevent multiple scans of data sets, and enhance memory usage. This significant contribution will be further discussed in the following chapter dedicated to experimentation and discussion. Table 14 provides the symbols used in this section and concise descriptions.

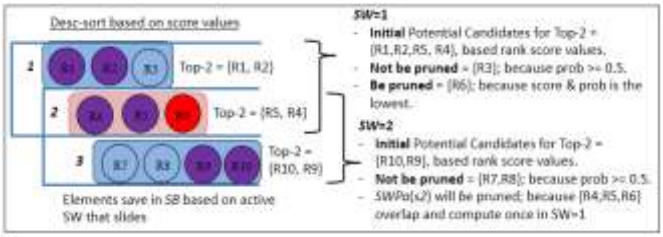


Fig. 11: The Initial sliding window frame with elements after $SWP^a(s_i)$ pruning

Table 14. Symbols used in this section and their explanations

Symbol	Description
UDS	Uncertain Data Stream
t_i	Uncertain tuple item / Tuple at timestamp i from UDS_i
s	Subsequence of stream from UDS_i
SW	Sliding Window Model
W	Window Size of |*SW*|
Score(t)	Score function of tuple item
Prob(t)	Probability function of tuple item
SWtop-k(t, S)	Probabilities cumulative of all possible worlds
p(w)	Possible world model rules where *w* is a collection
GM	Group membership
B_k	Bucket of Instance / tuple
BT_k	Bucket of potential candidates to be Top-k
SWP^a	Sliding window panes where partition-by attribute
$F_{disc} : T$	Discretization function on *Si*
$P(s[q])$	Top-k Potential Candidates
$SB(S_{ln})$	*Segmentation buffer*
Top-k Ψ	Top-k segmental sets queue
d	Indexes

6 Result and Discussion

An extensive experiment was conducted to evaluate the performance and efficacy of the *SWMTop-kDelta* framework. The algorithms and baseline approaches that have been compared are as follows:

i. The Top-k Incomplete Data Stream (*Topk-iDS*) technique, is a recent approach that utilizes a Count-based *SWM* to handle top-k queries in a possible world scenario.
ii. The Dominant Relationship Analysis (*DRA*) approach, determines the tuple items that are most likely to be the top-k results in query processing by minimizing the quantity of data. This technique does not rely on *SWM*.
iii. The summaries for directly adapting the current Count-based and Time-based sliding

windows prioritize recent data to generate a list of top-k potential candidates.

Two baseline algorithms can be used as reference points for comparing the framework:

i. The *Count*-based *SWM* determines the window's size based on the number of events it contains without a predetermined time duration for its validity. This method maintains the specified number of rows within the window frame. If a certain number of new points, denoted as *n*, are added, then the same number of the oldest points, also *n* in number, are removed.

ii. The *Time*-based *SWM* uses the constant number of active points. A point's termination time is independent of the reaching or termination of other points. Tuple items that have exceeded the specified duration will be eliminated from the window frame, regardless of whether new rows were received.

In addition, the *DRA* algorithm conducted comparisons across three (3) types of Top-k query processing (including *U-Topk*, *U-kRanks*, and *Pk-Topk*), and these entries could serve as enhanced or optimized solutions without relying on SWM. The *DRA* method is evaluated based on the number of possible world instances and their corresponding scores and probabilities. The *Topk-iDS* algorithm specifically deals with top-*k* queries on an incomplete and uncertain data stream. The data structure used for this purpose was top-k dual layers (*TDL*).

In the first phase, Phase I, we utilize the initial data to derive a group set of tuple items. The final top-k result list is generated by *Phase* II of the *SWMTop-kDelta* by comparing its result lists, *R (R')*, with the set of Top-k result lists produced by the previous methods, *Rp (Rp')* where these variables seem to represent the same *SWMTop-kDelta* result list. The experiment is conducted in five repetitions, and the resulting average value is recorded. The performance metrics utilized in our experiments are the expected *SWM* Top-k query result answer and processing time.

The measurements are evaluated for multiple parameter configurations, which include the size of the data set (K), the parameter (*k*), the window size (*S*), the probability threshold (*d*), and the K-Pruning (*r*). The sizes of the available datasets, including the synthetic data set, range from 5K (*Chicago Trip Taxi*) to over 200K (*AirBeijing*). The dimensions of the *Chicago Trip Taxi* and *AirBeijing* data sets are 5 and 9, respectively, while the dimension of the *AirBeijing* data set is 6. The value of this parameter is modified in multiple experiments, as outlined in Table 15.

Table 15. The parameter settings of the synthetic and real data sets

Parameter Settings	Data Sets		
	Synthetic	*Chicago Trip Taxi*	*AirBeijing*
Data Set Size (*n*)	5K, 20K, 50K, 100K, 200K	5K, 20K, 50K, 100K, 200K	5K, 20K, 50K, 100K, 200K plus
Parameter (*k*)	2, 4, 6, 8, 10	2, 4, 6, 8, 10	2, 4, 6, 8, 10
Window Size (*W*) & Segmentation	ranges from 10 to 10^4		
Probability Threshold (*d*)	0.3, 0.4, 0.6, 0.7, 0.8	0.3, 0.4, 0.6, 0.7, 0.8	0.3, 0.4, 0.6, 0.7, 0.8
Number Queries (*u*)	Each 10 secs until 1 minutes	Each 10 secs until 1 minutes	Each 10 secs until 1 minutes

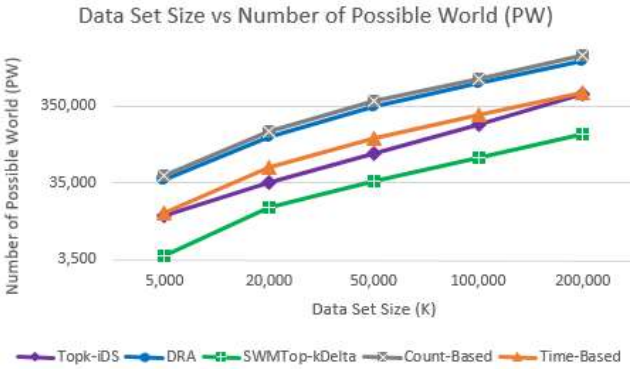
An evaluation was conducted on efficiency and scalability, focusing on the number of comparisons made between possible worlds and the processing time required. The top-k on a *UDS* is influenced by two parameters and compares the settings for the two measures. *Phase* I generates the initial set of candidate lists and collects other required information. The *Phase* II component is triggered based on the type of data modification to minimize overlapping computation and apply an insertion/exit policy to the relevant tuple candidates. Thus, *Phase* III builds upon the information that was obtained in *Phases* I and II by implementing optimization techniques to prevent unnecessary top-*k* computations when determining and computing possible world vector rules until the potential candidates expired from the window frame. We assume that lower checkpoint values on **Green** graft are better than higher ones when creating the collection of successful top-k final lists.

6.1 Effect of Data size

The results are present in Figures 12(a - c). As the number of tuple items in a data set increases, the number of possible world comparisons required to determine the validity of the data set also increases. When the size of data sets reaches 20K or more, it becomes evident that the *SWMTop-kDelta* algorithm outperforms the *Topk-iDS*, *DRA*, and baseline Count-based/Time-based *SWM* algorithm. The goal is to minimize redundant computation by implementing *SWMTop-kDelta* segments with an eviction policy delta. This approach aims to prevent the generation of unnecessary possible world rules. Using synthetic data, the *SWMTop-kDelta* method effectively decreases the number of potential world comparisons by 35% compared to the *Topk-iDS* approach. The subsequent step involves a decrease of 60% compared to Time-based SWM and a more than 95% reduction compared to Count-based SWM

and DRA algorithms. Possible world number comparisons involve comparing the items of a tuple that will be included in the future Top-k list of occurrences in a *SWM*.

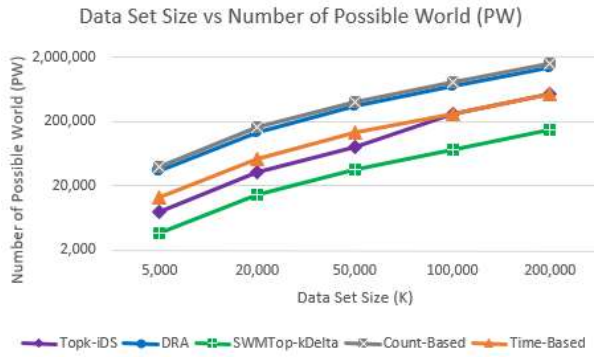
As the volume of the data set increases, the processing time for the mentioned algorithm will also increase. Our proposed *SWMTop-kDelta* algorithm was able to cut down on the number of possible worlds generated and based on these graft, it is clear that it has demonstrated superior performance by reducing overlapping computation segments using an eviction policy delta and avoiding generating unnecessary possible world rules. However, the main goal is to resolve the ambiguity caused by the uncertain data stream. Therefore, as the data set volume grows, our proposed algorithm maintains consistent performance, regardless of the data set size. Assuming the candidate's tuple has reached the first sliding window, the combined possible world is subjected to number comparisons with the same top-k tuple items. The number of possible worlds required was significantly reduced by implementing the following steps in *SWMTop-kDelta*. The analysis of the numbers indicates that *SWMTop-kDelta* consistently maintains its performance across data sets of varying sizes.



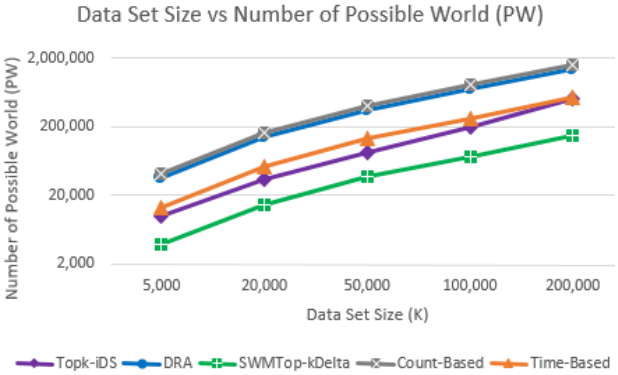
(c) *AirBeijing*

Fig. 12 (a - c): The results of possible world number comparisons with varying data set size

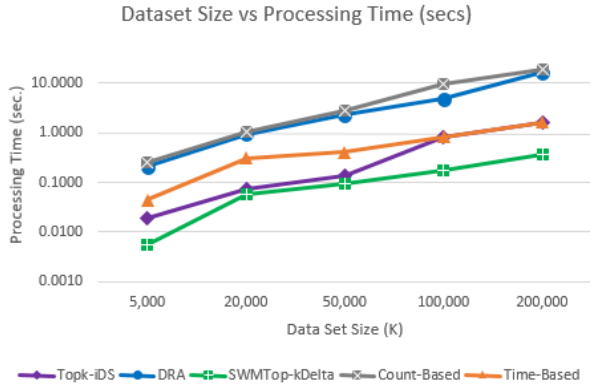
Figures 13 (a – c) present the reduction in the number of possible worlds leads to a decrease in the time required for query processing. Both the DRA and the baseline count-based *SWM* algorithm exhibit inadequate efficiency when comparing possible world numbers and processing time. The *Topk-iDS* algorithm is considered to be the closest performance rival to *SWMTop-kDelta*. The results demonstrate the percentage improvement of *SWMTop-kDelta* compared to the *Topk-iDS*, Time-based SWM, Count-based *SWM*, and *DRA* algorithms, as shown in Table 16.



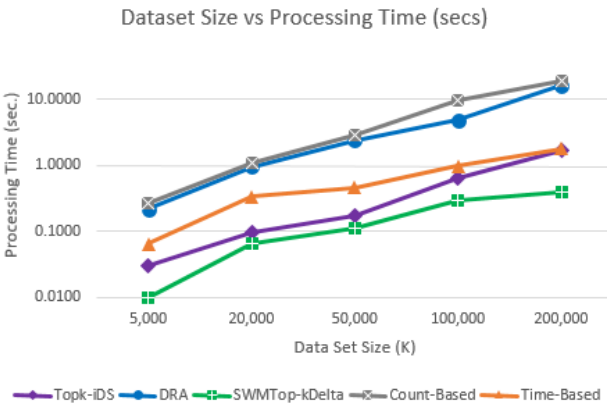
(a) Synthetic



(b) *Chicago Trip Taxi*



(a) Synthetic



(b) *Chicago Trip Taxi*

(c) *AirBeijing*

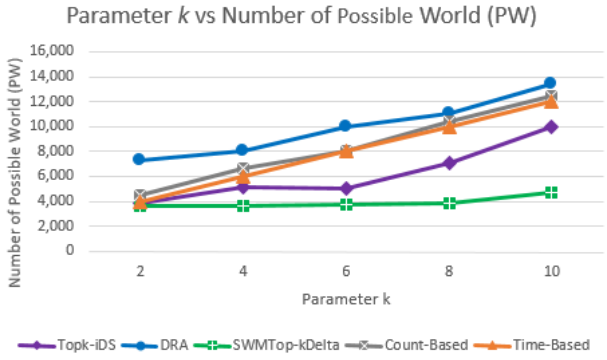
Fig. 13 (a - c): The results of processing time with varying data set size

Table 16. Percentage improvement of SWMTop-kDelta in terms of Data Set Size

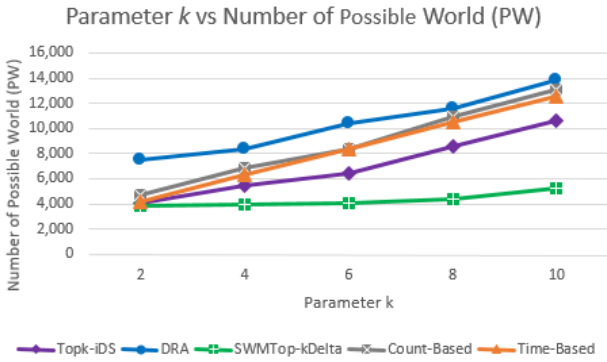
Algorithms compared	Dataset (independent)	Number of possible world comparisons	Processing time
Topk-iDS	Synthetic	61.68%	56.74%
	Chicago Trip Taxi	61.50%	51.76%
	AirBeijing	62.23%	33.76%
DRA	Synthetic	89.34%	96.26%
	Chicago Trip Taxi	89.35%	95.05%
	AirBeijing	89.06%	74.73%
Count-based	Synthetic	90.86%	97.07%
	Chicago Trip Taxi	90.85%	96.20%
	AirBeijing	90.52%	81.22%
Time-based	Synthetic	72.22%	80.51%
	Chicago Trip Taxi	71.86%	77.45%
	AirBeijing	71.33%	58.70%

6.2 Effect of Number of Parameter (*k*)

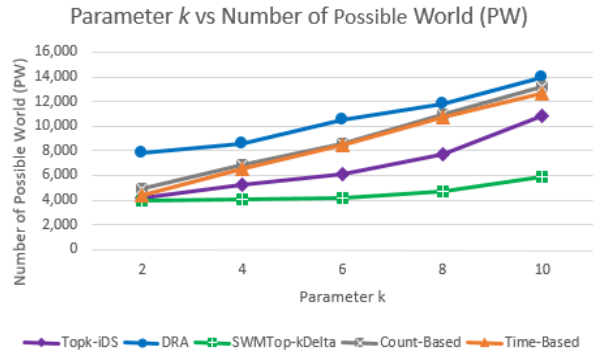
The objective is to evaluate the performance of *SWMTop-kDelta* in handling Top-k continuous queries over uncertain data streams with varying parameter values (*k*). We vary the parameter (*k*) values as 2, 4, 6, 8, and 10. In addition, instead of using the usual three score distributions $d \in u, n, e$ (uniform, normal, and exponential), only the uniform distribution $d \in u$ was chosen. The dimensions for the synthetic, *Chicago Trip Taxi*, and *AirBeijing* data sets are fixed at 5, 6, 6, and 9, respectively. Figures 14 (a - c) display the number of possible world comparisons that have been accomplished. . The results of the algorithms suggest that as the number of parameter (*k*) values increases, the number of conducted possible world comparisons also increases. This second comparison measurement is about the effect of Number Parameter k, where our proposed algorithm outperformed others regarding possible world number comparison because it remains a consistent performance and robust scalability, even where the number of parameter (k) values changes until it reaches 10 value.



(a) Synthetic



(b) *Chicago Trip Taxi*

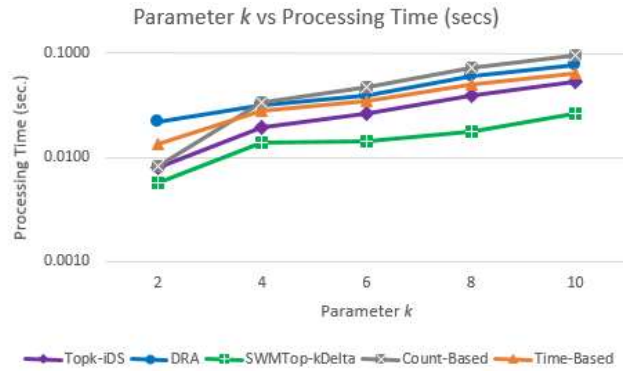


(c) *AirBeijing*

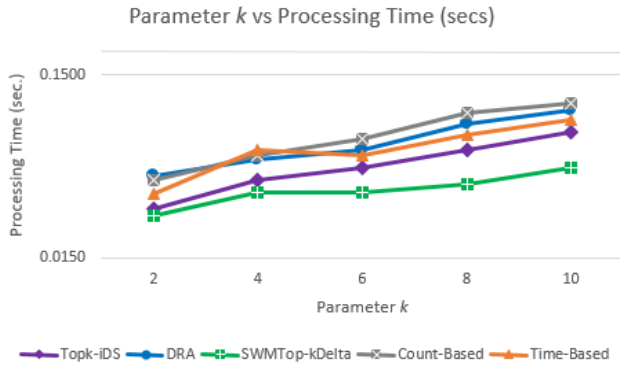
Fig. 14 (a - c): The results of possible world number comparisons with varying numbers of parameter (k) values

In terms of processing time, our algorithm outperformed others when the number of parameter (k) values increased through delta segmentation in the eviction policy. This is because the parameter k, which affects the top-10 values, takes longer to compute, resulting in more potential candidates being chosen. The processing time achieved with *SWMTop-kDelta* consistent performance across different parameter (k) values than *Topk-iDS*, *DRA*,

and the baseline count-based/time-based SWM algorithms. Figures 15 (a - c) exhibit comparable patterns from figures before, as a result of the reduction in the number of potential world comparisons, leading to a decrease in processing time. However, *SWMTop-kDelta* consistently achieves shorter processing times than other algorithms, such as *Topk-iDS*, *DRA*, and baseline count-based/time-based *SWM* algorithms.



(a) Synthetic



(b) *Chicago Trip Taxi*

(c) AirBeijing

Fig. 15 (a - c): The results of processing time with varying number of parameter (k) values

This is due to its use of tuple item comparison with pruning, which reduces the need to compute potential candidate lists from the specified sliding window. This approach is more efficient than comparing tuple items across the entire data set. The results of all algorithms demonstrate this relationship. As shown in Table 17, *SWMTop-kDelta* outperforms *Topk-iDS*, time-based SWM, count-based *SWM*, and *DRA* algorithms regarding an improvement percentage of over 30 % for both the number of possible world and processing time comparison.

Table 17. Percentage improvement of SWMTop-kDelta in terms of Number of Parameter (k)

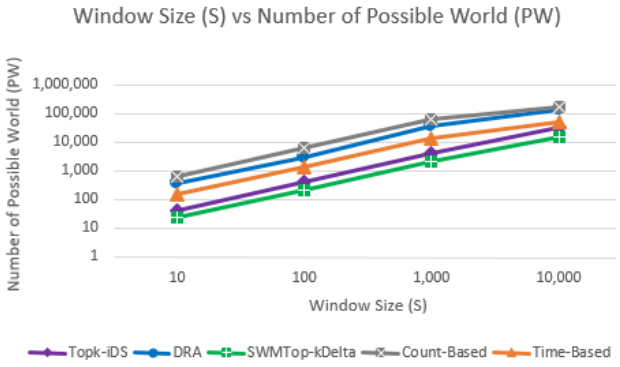
Algorithms compared	Dataset (independent)	Number of possible world comparisons	Processing time
Topk-iDS	Synthetic	31.93%	41.47%
	Chicago Trip Taxi	33.68%	24.07%
	AirBeijing	28.77%	29.23%
DRA	Synthetic	59.61%	65.78%
	Chicago Trip Taxi	57.29%	43.81%
	AirBeijing	56.08%	40.08%
Count-based	Synthetic	48.59%	61.33%
	Chicago Trip Taxi	46.38%	47.29%
	AirBeijing	44.83%	45.08%
Time-based	Synthetic	44.86%	57.95%
	Chicago Trip Taxi	31.93%	41.47%
	AirBeijing	33.68%	24.07%

6.3 Effect of Window Size (W) and Segmentation

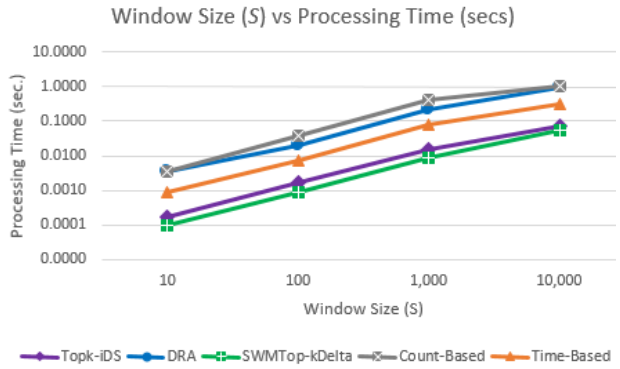
This study examines the impact of varying the window size (*W*) during segmentation on the effectiveness of *SWMTop-kDelta*. It can be inferred that the computation of window sizes can be reduced when more window segments are created. The eviction of a tuple item in this scenario is determined by the attribute delta, with a value of 10 seconds. This attribute enables a sliding window mechanism. To maintain the delta invariant, it is essential to calculate the combination of score and probability before *UDS* timestamp. The range of window sizes (*W*) is from 10 to 10^4.

By utilizing a dataset size of 20K and implementing a sliding window that covers a range of 10 segments to >= 2,000 segmentation units, significant improvements can be observed in the performance of *SWMTop-kDelta*. At this iteration, a higher amount of pruning is applied to the intra-window segmentation comparisons, while a lower amount of pruning is applied to the inter-window segmentation comparisons. It makes sense to assume that having fewer window sizes would result in more computational work when done in large quantities. This is because it would be necessary to construct a more significant number of window segments, which

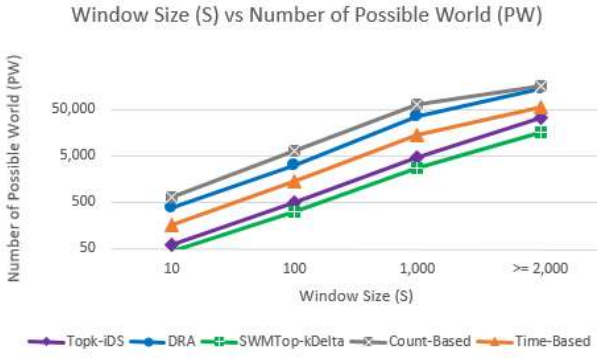
can restrict the computational possibilities within smaller segments.



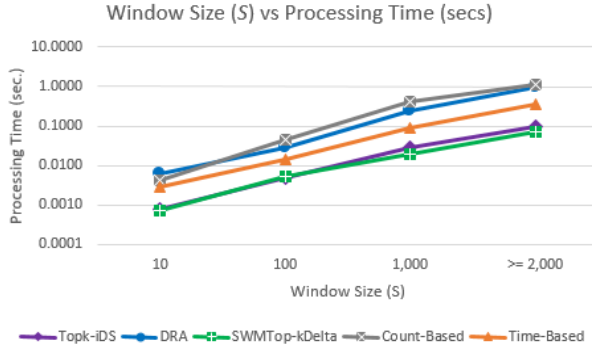
Number of Possible World



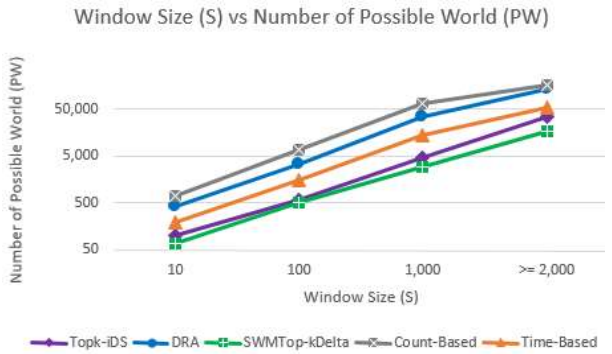
Processing Time
(a) Synthetic



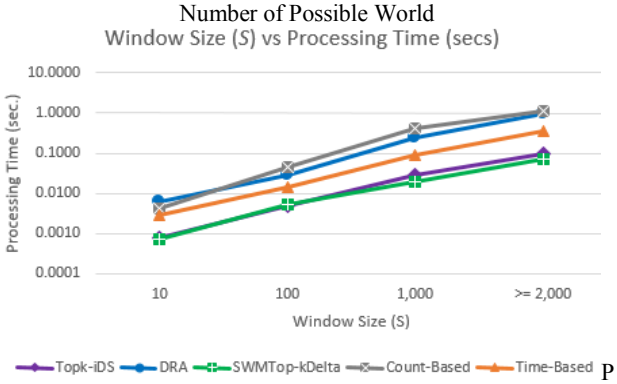
Number of Possible World



Processing Time
(b) Chicago Trip Taxi



Number of Possible World



Processing Time
(c) AirBeijing

Fig. 16 (a - c): The results of the number of possible world comparisons and processing time with varying numbers of Window Size (W) and Segmentation

Table 18. Percentage improvement of SWMTop-kDelta in terms of Window Size (W) and Segmentation

Algorithms compared	Dataset (independent)	Number of possible world comparisons	Processing time
Topk-iDS	Synthetic	46.45%	40.58%
	Chicago Trip Taxi	38.76%	25.41%
	AirBeijing	33.75%	17.13%
DRA	Synthetic	92.08%	95.83%
	Chicago Trip Taxi	89.77%	91.32%
	AirBeijing	87.22%	88.54%
Count-based	Synthetic	95.08%	96.96%
	Chicago Trip Taxi	93.51%	93.49%
	AirBeijing	91.99%	90.24
Time-based	Synthetic	80.63%	87.08%
	Chicago Trip Taxi	75.17%	76.86%
	AirBeijing	70.22%	74.44%

More pruning is done on greater intra-window segmentation and less on more extensive inter-window segmentation comparisons at this iteration. Whenever we increase the number of window segmentations while decreasing the window size, the performance remains intact and even improves slightly. Based on the analysis of Figure 16 (a-c) and

the data provided in Table 18, it can be concluded that *SWMTop-kDelta* outperforms *Topk-iDS*, Time-based *SWM*, Count-based *SWM*, and *DRA* algorithms regarding improvement percentage.

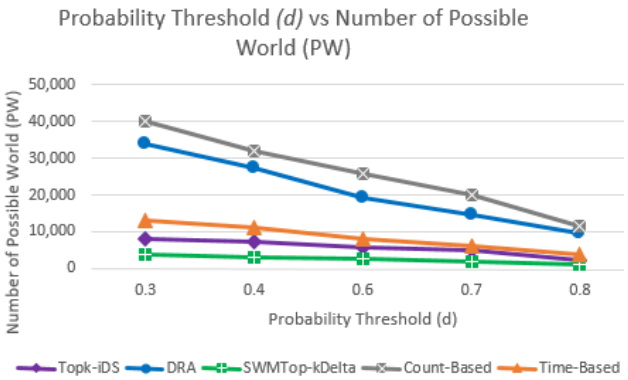
6.4 Effect of Probability Threshold (*d*)

The performance of top-k algorithms in handling continuous queries over *UDS* is greatly influenced by the frequency of changes to the probability threshold (*d*) and the state of the sliding window. The process includes adjusting the probability threshold (*d*) values to compute the number of possible world comparisons and processing time. The probability threshold rate (*d*) is set at 90% of the probability attribute from each dataset. The number of probability thresholds (*d*) varies in the following manner: the initial probability is set at 0.3, 0.4, 0.6, 0.7, and finally reaches 0.8. The presented numbers provide proof of the consistent performance of *SWMTop-kDelta* as shown in Figures 17 (a - c).

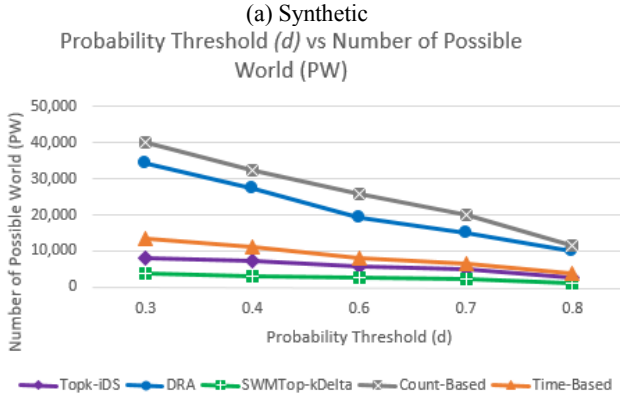
Possible world comparisons rise as the probability thresholds (d) decrease to 0.5 and below. The comparison of processing time is significantly impacted by the frequency of changes to the probability threshold and the status of the sliding window, as shown by the results of the previous algorithms. The *SWMTop-kDelta* algorithm has shown superior performance compared to *Topk-iDS*, *DRA*, and baseline count-based/time-based *SWM* algorithms. Despite its slight increase, *SWMTop-kDelta* outperforms these algorithms by eliminating unnecessary Top-k scores and probabilities computations.

Figures 18 (a-c) illustrate the strong performance of *SWMTop-kDelta* in reducing the distributions of minimal probability threshold (*d*) values within the sliding window. This reduction impacts the k possible answers while having minimal effect on the performance of *SWMTop-kDelta*. The processing time can be improved by reducing the number of possible world comparisons. The task involves calculating the precise probabilities that surpass the score threshold to generate potential candidate lists. This approach requires significantly less effort than comparing individual tuple items from the entire dataset against all possible world generation rules.

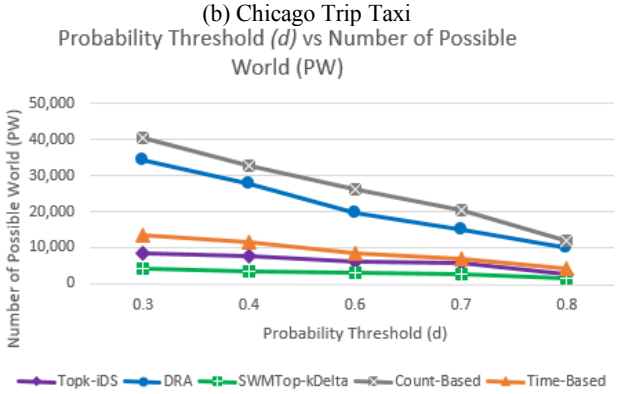
The *SWMTop-kDelta* algorithm outperforms the *Topk-iDS*, Time-based *SWM*, Count-based *SWM*, and *DRA* methods in terms of improvement percentage, as demonstrated in Table 19. Based on percentage improvement, *SWMTopk-Delta* reduces unnecessary computations by focusing on fewer possible world vector rules and optimizing sorting for remaining tuples using a sliding window approach based on a combination score and probability function.



(a) Synthetic

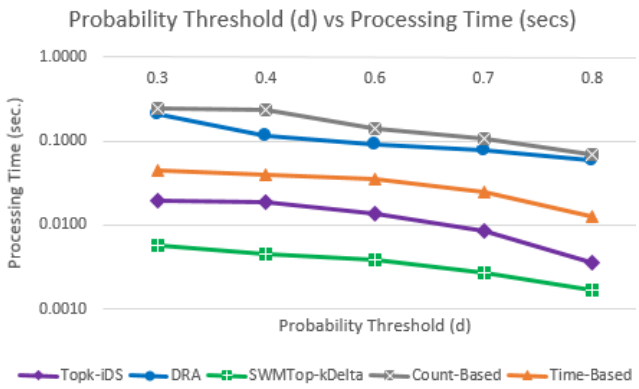


(b) Chicago Trip Taxi

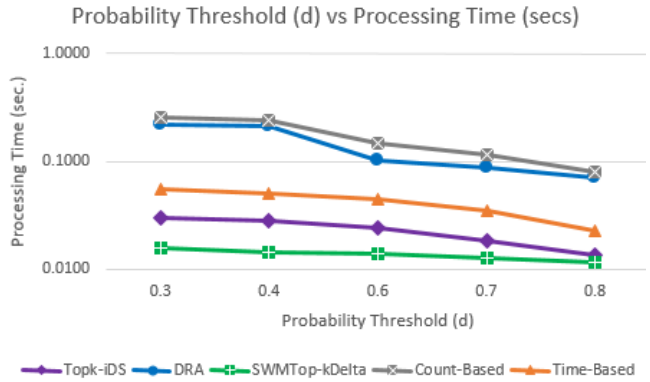


(c) AirBeijing

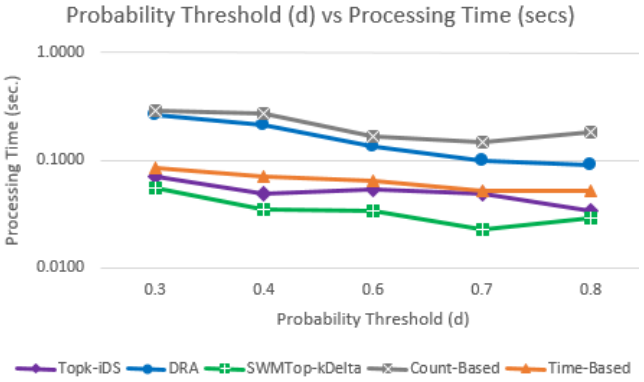
Fig. 17 (a - c): The results of the number of possible world comparisons with varying numbers of probability thresholds (d)



(a) Synthetic



(b) Chicago Trip Taxi



(c) AirBeijing

Fig. 18 (a - c): The results of processing time with varying numbers of probability thresholds (d)

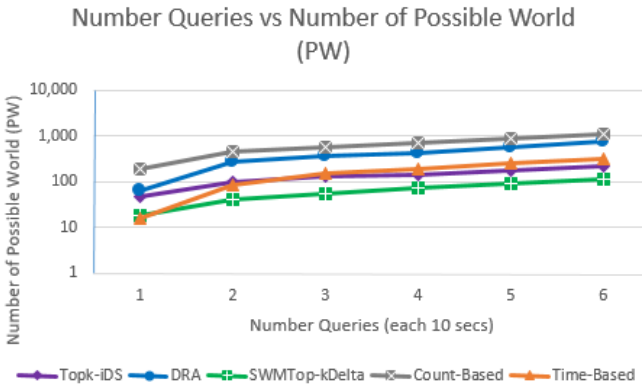
Table 19. Percentage improvement of SWMTop-kDelta in terms of Probability Threshold (*d*)

Algorithms compared	Dataset (independent)	Number of possible world comparisons	Processing time
Topk-iDS	Synthetic	56.82%	68.08%
	Chicago Trip Taxi	54.51	36.48%
	AirBeijing	49.99%	30.62%
DRA	Synthetic	88.23%	96.58%
	Chicago Trip Taxi	87.22%	88.17%
	AirBeijing	85.33%	76.46%
Count-based	Synthetic	90.54%	97.62%
	Chicago Trip Taxi	89.70%	90.59%
	AirBeijing	88.33%	83.22%
Time-based	Synthetic	70.68%	88.22%
	Chicago Trip Taxi	68.58%	64.59%
	AirBeijing	65.18%	46.81%

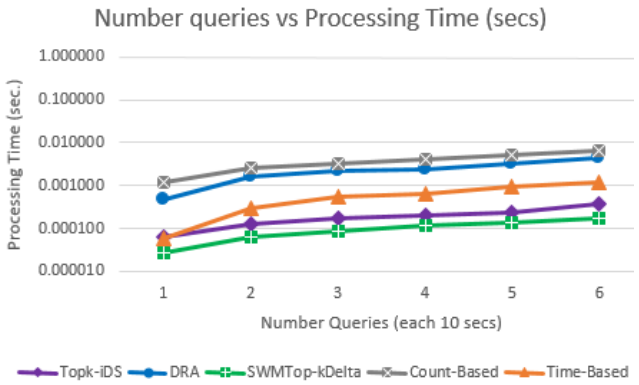
6.5 Effect of Number of Queries

We examined the impact of the execution query number on the performance of *SWMTop-kDelta*. The construction of highly correlated probabilistic streams, both physically and temporally, is directly proportional to the number of continuous queries performed. The queries are executed on data sets that initiate every 10 seconds and continue until they are complete. The parameter settings we employ are as follows: the probability threshold rate is configured to be 20% or higher, and the data sets for synthetic, *Chicago Trip Taxi*, and *AirBeijing* are each set to 5K. Figures 19 (a) demonstrate that the growth of all algorithms is linear as the number of single static queries *u* increases. When the number of queries reaches 20 seconds, there is a slight improvement in the performance of *SWMTop-kDelta*, *Topk-iDS*, and Time-based *SWM*. *SWMTop-kDelta* algorithm outperformed previous algorithm regarding the number of possible worlds and processing time. This is because the more continuous queries are performed, the more probabilistic streams highly correlated physically and temporally are constructed. Initially, it shows consistent performance with a steady increase as the number of queries grows. However, when the number of queries hits 20, the performance of *SWMTop-kDelta* surpasses the others.

Figure 19 (b) exhibits patterns comparable to Figure 19 (a), as the processing time demonstrates a slight increase at 20 seconds. This occurrence is because the number of possible worlds generated depends on the tuple items maintained throughout the continuous query time frame. Reducing query execution time directly leads to a decrease in processing time. The number of possible worlds generated for percentage improvement depends on the tuple items maintained during the continuous query time frame. If the query execution time is reduced, the processing time will also be reduced. Based on the findings presented in Table 20, it can be observed that the *SWMTop-kDelta* algorithm outperforms the *Topk-iDS*, the Time-based *SWM*, the Count-based SWM, and the *DRA* algorithm.

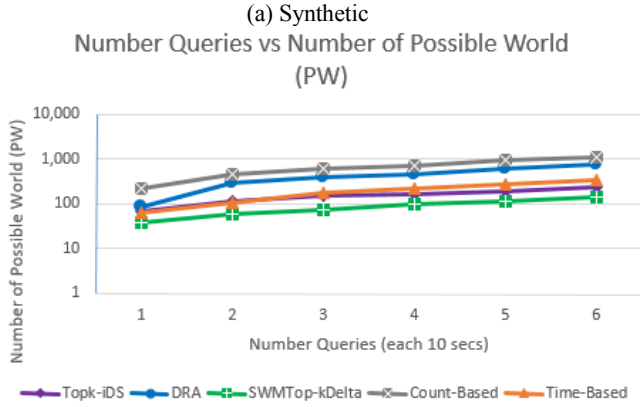


Number of Possible World

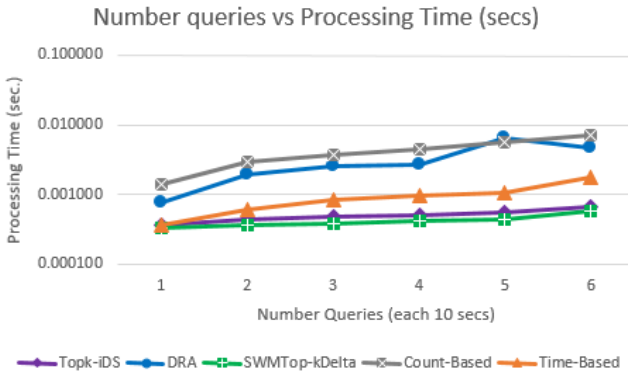


Processing Time

(a) Synthetic

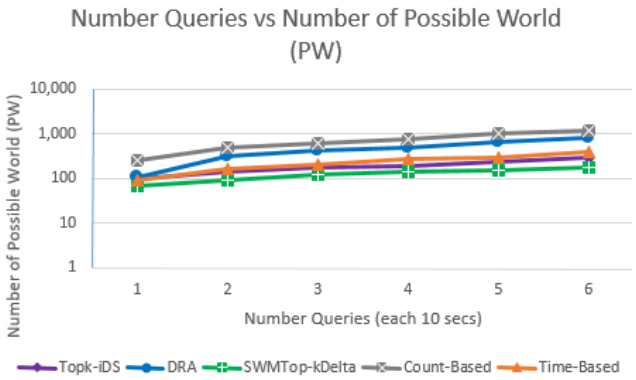


Number of Possible World

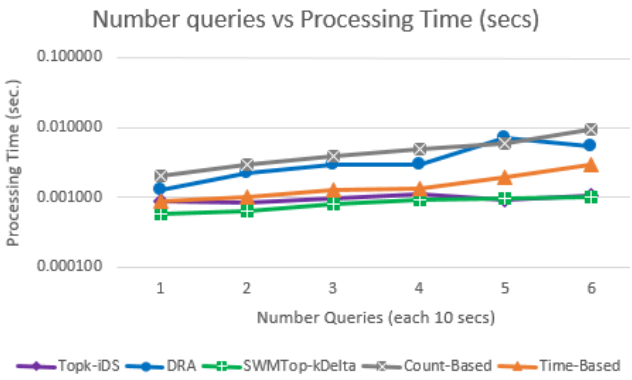


Processing Time

(b) Chicago Trip Taxi



Number of Possible World



Processing Time

(c) AirBeijing

Fig. 19: The results of the number of possible world comparisons and processing time with varying numbers of queries execution

Table 20. Percentage improvement of *SWMTop-kDelta* in terms of Number of Queries

Algorithms compared	Dataset (independent)	Number of possible world comparisons	Processing time
Topk-iDS	Synthetic	53.67%	49.03%
	Chicago Trip Taxi	44.52%	15.39%
	AirBeijing	33.75%	16.36%
DRA	Synthetic	82.22%	95.55%
	Chicago Trip Taxi	76.06%	81.33%
	AirBeijing	67.46%	72.58%
Count-based	Synthetic	90.10%	97.42%
	Chicago Trip Taxi	86.36%	87.97%
	AirBeijing	81.16%	80.55%
Time-based	Synthetic	49.50%	78.41%
	Chicago Trip Taxi	52.31%	47.74%
	AirBeijing	44.41%	42.70%

7 Conclusion and Future Work

Top-k queries are commonly employed in various critical applications to support data analysis and decision-making processes. This study presents the concept of tuple items probability theory and explores its application in creating anticipated rules for potential scenarios using a method involving *SWM*. We have already discussed the limitations of current approaches in dealing with UDS and semantic possibilities, and we will evaluate them alongside our proposed methods. The main goal of top-k queries is to provide users with relevant tuple items based on their preferences.

The research aims to propose an efficient top-k computation framework that can process continuous

queries on uncertain data streams over a *SWM*. We propose a solution, named *SWMTop-kDelta* framework, which consists of three main phases, namely: Phase I - aims to transform a collection of tuple items into a window fragment using the required *SWM* representation whenever the record vector tuple's necessary fragmentation is inserted, processed, and discarded, Phase II - executing an essential aspect of establishing the most efficient processing time using the proposed algorithm, which involves the complexity of top-k computation, which has a direct relationship starting with the continuous queries, score value, probability, timestamp interval (delta-attribute), and the possible world rules, which is exponentially correlated to the size of the data set, and Phase III - builds upon the information that was obtained in Phases I and II by implementing optimization techniques to prevent unnecessary top-k computations when determining and computing possible world vector rules until the potential candidates expired from the window frame. For each phase mentioned, a framework is proposed and designed.

Several evaluations have been conducted to evaluate the performance and efficiency of our proposed *SWMTop-kDelta* on uncertain data streams. The experiments employ performance metrics, such as comparisons of possible world numbers and processing time, across different parameter settings. These settings include data set size, number of parameters (k), window size (W) and segmentation, probability threshold (d), and number of queries. The study thoroughly investigated different scenarios, utilizing both synthetic and real data, to showcase the efficiency and performance of the proposed *SWMTop-kDelta* algorithm. Additionally, it surpasses the baseline *SWM* algorithm in terms of percentage improvement. The reported results are an improvement exhibited by *SWMTop-kDelta*, as evidenced in Table 16, Table 17, Table 18, Table 19 and Table 20, concerning the number of possible world comparisons. The analysis reveals a significant reduction of more than 61% in the generation of possible world rules. Consequently, there is an average improvement in processing time efficiency exceeding 55% across all the performance metrics. The research conducted in this study aims to reduce costs and enhance the efficiency of decision-making processes by enabling faster and more effective results. To demonstrate the effectiveness and significance of these studies, based on experimental results in contributing to the existing body of knowledge of query processing, we propose several vital suggestions:

Enhanced Real-Time Data Processing: Our proposed *SWMTop-kDelta* approach enables efficient handling of real-time data streams by avoiding frequent full recomputations. Through efficiency improvements, it allows for incremental updates, reducing computational overhead and improving response times compared to traditional top-k query processing methods. Efficiently processing high-velocity data streams is crucial for modern applications, as it allows for scalability and improved performance.

Handling Uncertainty: Demonstrating the effective handling of uncertainty when processing top-k queries through a combination of score and probability value computation. This will improve understanding of managing and disseminating uncertain data effectively, providing models and techniques that can be applied to other uncertain data processing tasks.

Algorithmic Innovations: In our research, we have developed and tested the *SWMTop-kDelta* algorithm, showcasing its effectiveness and making a valuable contribution to algorithm design. Our algorithm efficiently handles top-k queries within the context of *SWM* on uncertain data streams, outperforming previous research efforts.

Three (3) essential future research areas have been identified as suggestions for other researchers to pursue.

i. **Crowdsourcing** - In the context of the current diverse, complicated, and complex data management landscape, improving the efficiency and effectiveness of crowdsourcing data is of utmost importance. The presented approach offers a novel opportunity to incorporate human intelligence in addressing top-k queries.
ii. **Distributed Data Streams** – In the context of the current diverse, complicated, and complex landscape of data management, it is of utmost importance to improve the efficiency and effectiveness of crowdsourcing data. The presented approach offers a novel opportunity to incorporate human intelligence in addressing top-k queries.
iii. **Advanced uncertain data stream analysis in Cloud Environment** – The scenario presents a significant amount of data that is both extensive and intriguing. This is related to the availability of numerous regions for exploration and discovery, made possible by transmitting a large volume of data over the cloud environment. The investigation of processing large uncertain data streams and

incomplete and dynamic data on the cloud to derive an advantageous top-k result list for users is a noteworthy subject.

To ensure the reproducibility of future research studies, it is essential to provide other researchers with the specific details necessary to replicate our study. These details are as follows:

Presenting a comprehensive methodology explanation: *SWM* presents detailed explanations of algorithms and data structures. The Top-kDelta framework thoroughly explains top-k query processing models, encompassing delta-based *SWM* evaluations and score-probabilities values methods.

Open data and algorithm: We will ensure that all datasets used and source code can be shared on the repository website upon request, enabling other researchers to replicate our experiments. Comprehensive comments and documentation accompany the code to facilitate comprehension and utilization.

Parameter Settings: The parameter settings include factors such as the size of the data set, the number of parameters (k), the window size (W) and segmentation, the probability threshold (d), and the number of queries. These settings are crucial for accurately reproducing our experimental conditions. The number of possible world comparisons and processing time are also considered dependent factors. By providing detailed information about these settings, we enable others to recreate our experiments precisely.

Statistical Analysis: Thoroughly demonstrate the statistical analyses used to validate our outcome results, such as significance assessments and confidence intervals. This ensures the reliability and reproducibility of research findings, which can benefit future research aiming to achieve similar or improved results.

Adhering to these practices guarantees that our research on top-k query processing for uncertain data streams is transparent, replicable, and verifiable by the broader research community.

Declaration of Generative AI and AI-assisted Technologies in the Writing Process

During the preparation of this work the authors used ChatGPT in order to acquire some information about literature review and baseline technique example. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References:

[1] Ilyas, I. F., Beskales, G., & Soliman, M. A. (2008). A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4), 11:1-11:58. https://doi.org/10.1145/1391729.1391730.

[2] Fagin, R., Kumar, R., Sivakumar, D., Road, H., & Jose, S. (2003). Comparing top k lists. *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, San Jose, California, 1(2003), 134–160. https://doi.org/10.1137/S0895480102412856.

[3] Mingyi, D., & Yinju, L. (2015). An Effective Uncertain Data Streams Top-K Query Algorithm. *The Open Automation and Control Systems Journal*, 7(1), 1549–1553. https://doi.org/10.2174/1874444301507011549.

[4] Aggarwal, C. C., Member, S., & Yu, P. S. (2009). A Survey of Uncertain Data Algorithms and Applications. *IEEE Transactions on Knowledge and Data Engineering*, *21*(5), 609–623. https://doi.org/10.1109/TKDE.2008.190.

[5] Wahab, R. A. S. R., Mohd Rum, S. N., Ibrahim, H., Sidi, F., & Ishak, I. (2021). A Method for Processing Top-k Continuous Query on Uncertain Data Stream in Sliding Window Model. WSEAS Transactions on Systems and Control, 16, 261–269. https://doi.org/10.37394/23203.2021.16.22.

[6] Jin, C., Chen, L., Yu, J. X., & Lin, X. (2008). Sliding-Window Top-k Queries on Uncertain Streams. *VLDB 08*, (August 24-30), 24–30. https://doi.org/10.14778/1453856.1453892.

[7] Sarma, A. D., Benjelloun, O., Halevy, A., and Widom, J.:Working models for uncertain data, in *22nd International Conference on Data Engineering (ICDE'06)*, Atlanta, GA, USA, 2006, pp. 7-7. https://doi.org/10.1109/ICDE.2006.174.

[8] Ré, C., Letchner, J., Balazinksa, M., and Suciu, D., Event queries on correlated probabilistic streams, in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of data*, 2008, Vancouver, BC, Canada, pp. 715-728. https://doi.org/10.1145/1376616.1376688.

[9] Burdick, D., Deshpande, P., Jayram, T., Ramakrishnan, R., and Vaithyanathan, S: OLAP over uncertain and imprecise data, in *VLDB*, 2007, pp. 970-981. https://doi.org/10.1007/s00778-006-0033-y.

[10] Jiang, W., Wang, T., & Wang, Z. (2020). A Top-K Query Scheme with Privacy Preservation for Intelligent Vehicle Network in Mobile IoT. IEEE Access, 8, 81698–81710. https://doi.org/10.1109/ACCESS.2020.2990932.

[11] Fuhr, N. and Rölleke, T.: A probabilistic relational algebra for the integration of information retrieval and database systems, ACM Transactions on Information Systems (TOIS), vol. 15, pp. 32-66, 1997. https://doi.org/10.1145/239041.239045.

[12] Lakshmanan, L. V., Leone, N., Ross, R., and Subrahmanian, V. S.: Probview: A flexible probabilistic database system, ACM Transactions on Database Systems (TODS), vol. 22, pp. 419-469, 1997. https://doi.org/10.1145/261124.261131.

[13] Abiteboul, S., Kanellakis, P., and Grahne, G.: On the representation and querying of sets of possible worlds, *Theoretical computer science*, vol. 78, pp. 159-187, 1991. https://doi.org/10.1016/0304-3975(51)90007-2.

[14] Green, T.J., Tannen, V. (2006). Models for Incomplete and Probabilistic Information. In: Grust, T., Current Trends in Database Technology – EDBT 2006. EDBT 2006. *Lecture Notes in Computer Science*, vol 4254. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11896548_24.

[15] Li, L., Wang, H., Li, J., and Gao, H.: A survey of uncertain data management, Frontiers of Computer Science, vol. 14, pp. 162-190, 2020. https://doi.org/10.1007/s11704-017-7063-z.

[16] Carbone, P., Katsifodimos, A., & Haridi, S. (2019). Stream Window Aggregation Semantics and Optimization. Encyclopedia of Big Data Technologies,1615–1623. https://doi.org/10.1007/978-3-319-77525-8_154.

[17] Dallachiesa, M., Jacques-Silva, G., Gedik, B., Wu, K.-L., and Palpanas, T.: Sliding windows over uncertain data streams, *Knowledge and Information Systems*, vol. 45, pp. 159-190, 2015. https://doi.org/10.1007/s10115-014-0804-5.

[18] Chen, T., Chen, L., Oezsu, M. T., and Xiao, N.: Optimizing multi-top-k queries over uncertain data streams, *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, pp. 1814-1829, 2012. https://doi.org/10.1109/TKDE.2012.126.

[19] Minh, T., Le, N., Cao, J., & He, Z. (2013). Data & Knowledge Engineering Top-k best probability queries and semantics ranking properties on probabilistic databases. *Data & Knowledge Engineering*, *88*, 248–266. https://doi.org/10.1016/j.datak.2013.04.005.

[20] Agarwal, P. K., Sintos, S., & Steiger, A. (2020). Efficient Indexes for Diverse Top-k Range Queries. Proceedings of the 39*th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems. Association for Computing Machinery*, New York, NY, USA, 213–227. https://doi.org/10.1145/3375395.3387667.

[21] Gong, Z., Sun, G., Yuan, J., & Zhong, Y. (2009). Efficient Top- k Query Algorithms Using K -emSkyband. *Infoscale Journal*, 18, 288–305. https://doi.org/10.1007/978-3-642-10485-5_21.

[22] Ilyas, I. F., & Chang, K. C. (2007). Top- k Query Processing in Uncertain Databases. Proceedings of the *23rd International Conference on Data Engineering (ICDE 2007)*, Istanbul (Turkey), 896–905. https://doi.org/10.1109/ICDE.2007.367935.

[23] Khosla, C., & Kakkar, P. (2015). Top-k Query Processing Techniques in Uncertain Databases: A Review. *International Journal of Computer Applications*, 120(20), 33–37. https://doi.org/10.5120/21345-4358.

[24] Lin, S. (2010). *Rank aggregation methods*. John Wiley & Sons, Inc, 2(September/October 2010), 555–570. https://doi.org/10.1002/wics.111.

[25] Xiao, G., Li, K., Zhou, X., & Li, K. (2017). Journal of Computer and System Sciences Efficient monochromatic and bichromatic probabilistic reverse top- k query processing for uncertain big data. *Journal of Computer and System Sciences*, 89, 92–113. https://doi.org/10.1016/j.jcss.2016.05.010.

[26] Vlachou, A., Doulkeridis, C., Nørvåg, K., Vazirgiannis, M., Management, H. D., & Query, S. (2008). On Efficient Top-k Query Processing in Highly Distributed Environments. *Proceedings of the International Conference on Management of Data (ICMD08)*, Vancouver (Canada), 753–764. https://doi.org/10.1145/1376616.1376692.

[27] Xie, M., & Wood, P. T. (2013). Efficient Top-k Query Answering using Cached Views. *ACM Journal*, (March 18-22), 489–

500. https://doi.org/10.1145/2452376.2452433.

[28] Zhao, K., Tao, Y., & Zhou, S. (2007). Efficient top- k processing in large-scaled distributed environments. *Data & Knowledge Engineering*, 63, 315–335. https://doi.org/10.1016/j.datak.2007.03.012.

[29] M. Hua, J. Pei, W. Zhang and X. Lin, "Efficiently Answering Probabilistic Threshold Top-k Queries on Uncertain Data," *2008 IEEE 24th International Conference on Data Engineering*, Cancun, Mexico, 2008, pp. 1403-1405, doi: 10.1109/ICDE.2008.4497570.

[30] Bousnina, F. E., Chebbah, M., Anis, M., Tobji, B., Hadjali, A., & Yaghlane, B. Ben. (2017). On Top-k Queries over Evidential Data. *Proceedings Ofthe 19th International Conference on Enterprise Information Systems (ICEIS 2017)*, 1(Iceis), Porto, Portugal, 106–113. https://doi.org/10.5220/0006317701060113.

[31] Ge, S., U, L. H., Mamoulis, N., & Cheung, D. W. (2013). Efficient All Top- k Computation - A Unified Solution for All Top- k, Reverse Top- k and Top- m Influential Queries. *IEEE Transactions on Knowledge and Data Engineering*, 25(5), 1015–1027. https://doi.org/10.1109/TKDE.2012.34.

[32] Ge, T., Zdonik, S., & Madden, S. (2009). Top-k queries on uncertain data. Proceedings of the *2009 ACM SIGMOD International Conference on Management of Data, Providence,* Rhode Island USA, 375–388. https://doi.org/10.1145/1559845.1559886.

[33] Lin, C., Lu, J., Wei, Z., Wang, J., & Xiao, X. (2017). Optimal algorithms for selecting top-k combinations of attributes: theory and applications. *The VLDB Journal*, 27, 27–52. https://doi.org/10.1007/s00778-017-0485-2.

[34] Papadopoulos, A. N., Tiakas, E., Tzouramanis, T., Georgiadis, N., & Manolopoulos, Y. (2021). *Top-k Dominating Queries BT - Skylines and Other Dominance-Based Queries*. Skylines and Other Dominance-Based Queries, Synthesis Lectures on Data Management ((SLDM)), pp. 63-90. https://doi.org/10.1007/978-3-031-01876-3_4.

[35] Zhang, Z., Xie, X., & Pan, H. (2018). An Efficient Optimization Approach for Top-k Queries on Uncertain Data. *International Journal of Cooperative Information Systems*, 27(01), 1741002. https://doi.org/10.1142/S0218843017410027.

[36] Chen, J., & Feng, L. (2017). Efficient pruning for top-k ranking queries on attribute-wise uncertain datasets. *Journal of Intelligent Information Systems*, 48(1), 215–242. https://doi.org/10.1007/s10844-016-0403-x.

[37] Dai, C., Chen, L., & Chen, Y. (2012). An efficient algorithm for top-k queries on uncertain data streams. *2012 11th International Conference on Machine Learning and Applications*, Boca Raton, FL, USA, 19, 294-299. https://doi.org/10.1109/ICMLA.2012.57.

[38] Jiang, H., Zhu, R., & Wang, B. (2020). EPF: A General Framework for Supporting Continuous Top-k Queries over Streaming Data. *Cognition Computing*, 12, 176–194. https://doi.org/10.1007/s12559-019-09661-z.

[39] Liu, H., Zhou, K., Zhao, P., & Yao, S. (2018). Mining frequent itemsets over uncertain data streams. *Int. J. High Performance Computing and Networking*, 11(4), 312–321. https://doi.org/10.1504/IJHPCN.2018.093234

[40] Ren, W., Lian, X., & Ghazinour, K. (2021). Effective and efficient top- k query processing over incomplete data streams. *Information Sciences*, 544, 343–371. https://doi.org/10.1016/j.ins.2020.08.011.

[41] Shen, Z., Cheema, M. A., Lin, X., Zhang, W., & Wang, H. (2014). A Generic Framework for Top- k Pairs and Top- k Objects Queries over Sliding Windows. *IEEE Transactions on Knowledge and Data Engineering*, 26(6), 1349-1366. https://doi.org/10.1109/TKDE.2012.181.

[42] Xiao, N., Chen, T., Chen, L., & Tamer, M. O. (2013). Optimizing Multi-Top-k Queries over Uncertain Data Streams. *Transactions on Knowledge and Data Engineering*, 25(8), 1814–1829. https://doi.org/10.1109/TKDE.2012.126.

[43] Zarko, I. P., & Zi, P. (2015). Time- and Space-Efficient Sliding Window Top-k Query Processsing. *ACM Transactions on Database Systems*, 40(1), 1–44. https://doi.org/10.1145/2736701.

[44] Cormode, G., & Li, F. (2009). Semantics of Ranking Queries for Probabilistic Data and Expected Ranks. *2009 IEEE 25th International Conference on Data Engineering*, Shanghai, China, 16, 305-316. https://doi.org/10.1109/ICDE.2009.75.

[45] Rai, N., & Lian, X. (2023). Distributed probabilistic top-k dominating queries over uncertain databases. *Knowledge and*

Information Systems, 65(11), 4939–4965. https://doi.org/10.1007/s10115-023-01917-3.

[46] Li, L., & Wang, H. (2020). A survey of uncertain data management. *Frontiers of Computer Science*, 14(1), 162–190. https://doi.org/10.1007/s11704-017-7063-z.

[47] Wang, Y., Li, X., Li, X., & Wang, Y. (2013). A survey of queries over uncertain data. *Knowledge and Information Systems*, 37(3), 485–530. https://doi.org/10.1007/s10115-013-0638-6.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

Dr. Siti Nurulain Mohd Rum served as my advisor and mentor. I am extremely grateful to her for all of the hard work, invaluable research skills, helpful support, and valuable criticism that she provided. In addition, I would like to express my appreciation to Professor Dr. Hamidah Ibrahim and Associate Professor TS. Dr. Iskandar Ishak, both of whom were members of our advisory group, for the insightful criticism, positive reinforcement, smart counsel, and creative suggestions that they provided.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

There are no sources of Funding from me

Conflict of Interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.