Weighted Reservoir Sampling On Evolving Streams: A Sampling Algorithmic Framework For Stream Event Identification

Christos Karras
CEID
University of Patras
Patras, Rion, Greece
c.karras@ceid.upatras.gr

Aristeidis Karras
CEID
University of Patras
Patras, Rion, Greece
akarras@ceid.upatras.gr

Georgios Drakopoulos
Department of Informatics
Ionian University
Corfu, Greece
c16drak@ionio.gr

Konstantinos Tsakalidis
Department of Computer Science
University of Liverpool
Liverpool, Merseyside, UK
k.tsakalidis@liverpool.ac.uk

Phivos Mylonas
Department of Informatics
Ionian University
Corfu, Greece
fmylonas@ionio.gr

Spyros Sioutas
CEID
University of Patras
Patras, Rion, Greece
sioutas@ceid.upatras.gr

ABSTRACT

Data streams are becoming increasingly important across a wide array of fields and are generally expected to be the preferred form of big data as aggregators and smart stream analytics in general can efficiently yield stream descriptions in various levels. Among them, event detection analytics are paramount since they typically allow the identification of distinct cases of interest like the so called black swans. Reservoir sampling refers to probabilistic class of techniques for keeping representative values of a stream given limited memory capacity. In the proposed framework event detection takes place once reservoir sampling is complete by clustering its output. The rationale behind this is that repeated representative values correspond to normal stream states, whereas any outliers indicate rare yet noteworthy events. With that information a probabilistic stream state graph can be constructed in order to examine the transition dynamics between states and to evaluate the role black swans play in the overall stream stability. A major part of the descriptive power of said graph lies on its inherent geometric interpretation in addition to the algebraic one. Results from two benchmark datasets, one coming from real world and a random one, are encouraging. The proposed framework is planned to be executed in Raspberry Pi as part of an IoT stack since it is sufficiently lightweight.

CCS CONCEPTS

• **Information systems**; • **Mathematics of computing** → **Probability and statistics**;

KEYWORDS

reservoir sampling, data streams, geometric analytics, clustering, outlier discovery, probabilistic state graphs, black swan, IoT

SETN 2022, September 7–9, 2022, Corfu, Greece

ACM ISBN 978-1-4503-9597-7/22/09…$15.00
https://doi.org/10.1145/3549737.3549767

ACM Reference Format:
Christos Karras, Aristeidis Karras, Georgios Drakopoulos, Konstantinos Tsakalidis, Phivos Mylonas, and Spyros Sioutas. 2022. Weighted Reservoir Sampling On Evolving Streams: A Sampling Algorithmic Framework For Stream Event Identification. In *12th Hellenic Conference on Artificial Intelligence (SETN 2022), September 7–9, 2022, Corfu, Greece.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3549737.3549767

1 INTRODUCTION

Data streams are among the fundamental notions in the big data field as in an overwhelming number of engineering scenarios, especially in the context of smart cities, data collection and broadcast over streams from a plethora of heterogeneous IoT devices is standard practice in the field. Although streams are operationally quite less than tractable as a rule since they contain data to be mined and utilized within a short time frame, tailored stream analytics including among others aggregators and adaptive schemes capture at least the main characteristics of the underlying stream while maintaining an acceptable level of computational complexity.

Reservoir sampling is a class of probabilistic methodologies designed for identifying and storing significant stream values under the hard constraint of limited available memory capacity, translating to a tight correctness requirement for any decision algorithm designating stream values as important and progressively creating a stream summary. However, there is no distinction as to whether these values signify a normal condition or state of the underlying steam or they are *black swans*, namely rarely occurring events deserving special handling, nor do they reveal transition dynamics. The latter fact is the principal motivation behind this work.

The primary research objective of this paper is a three-phase framework for constructing a probabilistic state graph summarizing a stream. Specifically, once reservoir sampling is complete, the resulting values are clustered with outliers deemed as singular events, whereas in the final stage said graph is created based on transition dynamics. Besides attaining reasonable space and time complexity, the proposed framework has increased descriptive power driven from both probability theory and geometry, given that the shape of each individual cluster as well as the stream state graph topology

play a major role. This not only does pave the way for advanced geometric stream analytics, including heuristics, but also differentiate this work from most previous approaches.

The remainder of this paper is structured as follows. In section 2 the recent scientific literature regarding reservoir sampling, data streams, and geometric analytics is briefly overviewed. The proposed framework is described in detail in section 3, whereas the results obtained from the experiments are presented in section 4. Future research directions are given in 5. Technical acronyms are explained the first time they are encountered in the text. Finally in table 1 the notation of this work is summarized.

Table 1: Notation of this paper.

Symbol	Meaning	First in
$\triangleq$	Definition or equality by definition	Eq. (1)
$\{s_1, \ldots, s_n\}$	Set with elements $s_1, \ldots, s_n$	Eq. (1)
$\|S\|$	Set cardinality functional	Alg. 1
$S_1 \setminus S_2$	Asymmetric set difference	Alg. 1
prob $\{\Omega\}$	Probability of event Ω occurring	Eq. (7)
$\langle p_1 \,\|\| \, p_2 \rangle$	Kullback-Leibler divergence	Eq. (20)
$\|\mathbf{x}\|_2$	Euclidean norm	Eq. (13)

2 PREVIOUS WORK

Sampling across distributions is a significant concept in statistics, probability, systems engineering, and other fields that make use of stochastic models ([7, 33, 35, 38]). Markov Chain Monte Carlo (MCMC) techniques [25, 26] are often utilized to generate samples that closely approximate a given multivariate probability distribution. In particular, MCMC employs repeated random sampling to exploit the law of large numbers. Samples are generated by running a Markov Chain, which is created such that its stationary distribution follows the input function, for which a proposal distribution is used. It is feasible to decrypt encrypted documents [11], optimize functions [36], estimate integrals using generalized liner mixed models, and discover approximate solutions to non-deterministic polynomial-time (NP) hard problems using the Metropolis-Hastings algorithm. Markov Chain sampling schemes are introduced in [31] along with different possible combinations for sampling. Additionally, MCMC methods can be utilized for parameter estimation [32] while weighted stochastic versions of MCMC methods are introduced in [16].

Mining complex data streams for non-trivial knowledge is challenging [34]. It can take a multitude of forms such as regular event repetition discovery [8]. Window models are a viable alternative [22] with landmark window [13], sliding window [6, 41], and damped window [39] being the main approaches. Another strategy is to acquire representative samples with reservoir sampling [17, 28]. Another approach is to use neural networks for sampling user and item reviews to construct recommendation systems [12, 23].

Clustering and outlier detection have been widely studied in the recent years [5, 14, 29, 42]. The most frequently applied technique on clustering is k-means. Subsequent improvements include estimates of the number of clusters through k-median [3] or unlabeled multiclass SVMs [45]. STREAM relies on the notion of data distribution [19]. CluStream was designed to exploit information obtained by progressive clustering [2]. Due to the large number of clusters that must be created or manually picked at each phase human supervision is required [2]. As per [1], streamKM++ is a stream processing clustering methodology that is the foundation of the work in [4]. Cluster recovery can be done using data discrepancies that are specific to them [20] while tracking patterns in real time is proposed in [10].

Graph mining is a major driver across fields such as graph databases [15], Industry 4.0 process mining, social media analysis [9], logistics [21], and human omics [43, 44]. A related field is that of geometric analytics have found their way to distributed implementations such as the computational geometry pipelines for Hadoop in [30]. Moreover, frameworks for computational geometry have been introduced in MapReduce [18] while sampling across complex geometric objects is outlined in [24]. Visual analytics as well as heuristics for spatial clustering are presented in [37]. Finally, the pre-print version of this work can be found in [27].

3 METHODOLOGY

3.1 Overview

In this section the proposed framework is described for the particular case where the k-means algorithm with the Euclidean norm as distance metric for illustration purposes. Said framework operates along a line of reasoning where outliers or black swans are unique and exceptional events and consists of the following distinct steps:

- Reservoir sampling selects representative stream values.
- Clustering discovers outliers and unique events.
- The stream state graph is constructed.

The proposed framework is lightweight enough to be implemented on Raspberry pi as part of an IoT operations stack.

3.2 Reservoir Sampling

Assume the n items x_i being streamed come are drawn from a countable population set V as shown in equation (1) according to a fixed and known distribution. Moreover, to each x_i corresponds a not necessarily unique and known weight w_i.

$$V \triangleq \{x_1, \ldots, x_n\} \tag{1}$$

Random sampling without replacement requires selecting k distinct random items from a population of n individuals and equally distributing them among them. When all items have an equal probability of being picked, this is referred to as homogeneous random sampling. Weighted random sampling is a strategy whereabouts items are weighed, and the probability of any item being picked is indicated by the relative weight of the sample.

Algorithm 1 performs weighted random sampling without replacement. In particular let $\pi_j\,(i)$ denote the probability of selecting the j-th item in $V \setminus S$ in the t-th round

$$\pi_j\,(t) \triangleq \frac{w_j}{\sum_{s_t \in V \setminus S} w_t} \tag{2}$$

The algorithm 2 performs non-replacement weighted random sampling in line with the approach of the algorithm 1.

Proof. Consider the case when $k = 1$. Calculate the likelihood that the first item will be chosen. This necessitates the satisfaction

Algorithm 1 Weight Based Sampling

Require: A set V of n weighted items.
Ensure: Obtain $S \subset V$ such that $|S| = k$.
1: For values ranging from $t = 1$ to k
2: Assign $\pi_i(t)$ as in (2).
3: Choose a random item v_j from $V \setminus S$ and insert it into the S.

Algorithm 2 Weight Based Random Sampling

Require: A set V of n weighted items.
Ensure: Ascertain the existence of $S \subset V$ such that $|S| = k$.
1: xi is a sample from an unbiased uniform distribution spanning the interval $[0, 1]$, and the key for each v_i in V is specified as $k_i = x_i^{1/w_i}$.
2: x_i is a sample from an unbiased uniform distribution spanning the interval $[0, 1]$, and the key for each v_i in V is specified as $k_i = xi^{1/w_i}$.

of the following inequality:

$$x_1^{1/w_1} > \max\left[x_i^{1/w_i}\right], \qquad 2 \le i \le n \tag{3}$$

Assume that p_i is the likelihood that the very first element is chosen whenever condition (4) is satisfied:

$$i = \operatorname{argmax}\left[x_j^{1/w_j}\right], \qquad 2 \le j \le n \tag{4}$$

As a result, we have:

$$p_i \triangleq \int_0^1 \left(1 - x^{w_1/w_i}\right) x^{\frac{\sum_{i,\hat{1}}^{w_j}}{w_i}} dx \tag{5}$$

In (5) the following shorthand notation is used:

$$\sum_{\hat{i},\hat{j}} \triangleq 1 + \ldots + i - 1 + i + 1 + \ldots + j - 1 + j + 1 + \ldots + n \tag{6}$$

As a result, $p = \sum_{\hat{1}} p_i$ is the probability of such scenario.

The equation (5) is derived as follows. Let c_0 be a constant value for the sample x_i. Then the condition (7) holds with probability:

$$\text{prob}\left\{x_1^{1/w_1} > c_0^{1/w_i}\right\} = 1 - c_0^{w_1/w_i} \tag{7}$$

Similarly, the condition (8) holds with probability:

$$\text{prob}\left\{x_j^{1/w_j} < c_0^{1/w_i}\right\} = c_0^{w_j/w_i}, \quad \forall j, 2 \le j \ne i \le n \tag{8}$$

Then, since the samples are generated stochastically independently, the likelihood of any scenario can be computed by multiplying the above probabilities and integrating over the probability space of the samples x_i. The latter is the standard interval $I_0 = [0, 1]$ with constant density function equal to one.

We obtain the following by solving the integral from (5) :

$$p_i \triangleq \frac{1}{1 + \frac{\sum_{i,1} w_j}{w_i}} - \frac{1}{1 + \frac{\sum_i w_j}{w_i}} = \frac{w_i}{\sum_{\hat{1}} w_j} - \frac{w_i}{\sum w_j} \tag{9}$$

Immediately it follows that:

$$p = \sum_{\hat{1}} p_i = \frac{\sum_{\hat{1}} w_i}{\sum_{\hat{1}} w_j} - \frac{\sum_{\hat{1}} w_i}{\sum_{j=1}^{n} w_j} = \frac{w_i}{\sum_{j=1}^{n} w_j} \tag{10}$$

This outcome is identical to the second phase of algorithm 1, under the critical assumption that in each round the item with the largest key in $V \setminus S$ will be picked. □

Algorithm 3 is the reservoir sampling variant directly derived from algorithm 2. It should be noted that in steps 2 and 5 of algorithm 3 the key k_i for each random sample x_i drawn from an unbiased uniform distribution over I_0 which is weighted by w_i is computed as shown in equation (11):

$$k_i \triangleq x_i^{1/w_i} \tag{11}$$

As a sidenote, in some computational models the length of numbers in bits is a significant parameter determining the complexity of elementary operations. The length l_i of k_i is then given in (12):

$$l_i \triangleq \lceil \log_2 k_i \rceil = \left\lceil \frac{\log_2 x_i}{w_i} \right\rceil \tag{12}$$

Algorithm 3 Weight Based Reservoir Sampling

Require: A set V of n weighted items and reservoir R.
Ensure: Ensure that exists a $S \subset V$ such that $|S| = k$.
1: Initialize the R with the first k entries of V.
2: Compute key k_i of $x_i \in R$ as in (11).
3: For any $v_i \in V \setminus R$, do the following:
4: Let $k_m \in R$ be the smallest key.
5: Calculate the key for v_i as in (11).
6: If $k_i > k_m$, swap the x_i and x_m.

The algorithms 2 and 3 are equivalent because the algorithm 3 ensures that the items with the greatest k keys remain in the reservoir, either because they were originally began to be processed or because they will eventually be exchanged for an item with a lower key value.

3.3 Data Stream Clustering

The distance metric used in the context of this work for the k-means scheme was the Euclidean norm given in equation (13):

$$g(x_i, x_j) \triangleq \|x_i - x_j\|_2 \tag{13}$$

The geometric interpretation of (13) is that when the Euclidean distance is constant and equal to r_0, then x_i moves along the circumference of the circle whose center is x_j and its radius is r_0. Moreover, the Euclidean norm is invariant to linear translations and matrix unitary transforms but sensitive to scaling.

Apart from the conventional stages that are analogous to the class k-means method, the initial execution of the algorithm specifies the positions of k cluster centers premised on the positions of the k arbitrarily defined medoids, that is, on the spots of k database entries, in contrast to the conventional stages that are analogous to the class k-means mechanisms. As a result of performing the k-means procedure, we are able to maintain the resulting cluster centres for each cluster as well as the collection of samples, the Sum Squared Error (SSE) of each dimension of the individual clusters, and the SSE of the overall clustering. As a consequence, the items are eliminated to make room for new ones, allowing for the restart of the clustering process. The SSE [40] statistic is a frequently used

statistic for evaluating clustering techniques. The equation (14) is used to determine the SSE of a group.

$$\mathrm{SSE}_{\mathrm{group}} = \sum_{x \in C_i} \mathrm{g}(x, m_i)^2 \tag{14}$$

An element in the cluster C_i is denoted by x, and the centroid of a group is denoted by m_i. Due to the nature of clustering, the SSE of the whole clustering is defined as the summation of all k SSE values discovered in the subgroups, as seen in equation (15).

$$\mathrm{SSE}_{\mathrm{total}} \triangleq \sum_{j=1}^{k} \sum_{x \in C_i} \mathrm{g}(x, m_i)^2 \tag{15}$$

Each dimension of a feature in our application has its own SSE, which is computed separately. For instance, while dealing with 2D data, we maintain the SSE for each group, as defined in Equation (14), as well as the SSE for each 1D and 2D, as provided in Equation (14). The SSE of a single dimension is calculated using the equation (16), as shown in the following expression. The width of a vector t divided by the sum of the squared differences among distances between a point x and its cluster centre m_i. This computation is carried out for each of the data set's d dimensions.

$$\mathrm{SSE}_{\mathrm{dim}}(t) = \sum_{x \in C_t} \mathrm{g}\left(x_t, m_{i_t}\right)^2 \quad \forall \quad t \in d \tag{16}$$

Equation (16) is paramount as it is indicative of the overall clustering performance. In general, the lower the standard deviation of a group, the more tightly distributed the points are, and the better the group. The SSE is also computed for each dimension since it enables us to identify whether or not the data is more equally distributed in that dimension. Apart from the SSE, a number of additional cluster quality indicators are provided, including homogeneity, group density, intragroup maximum and lowest, and a variety of others. With the second cluster of the data stream, it seems as if the baseline centroids are no longer fully random. This approach will use the final centroids locations from the previous clustering phase. This prevents the simultaneous exchange of locations between two or more centroids. This enables visualisation of the development of a particular cluster throughout the course of a data stream, for example.

3.4 Probabilistic Graph Representation

In this subsection the final step of the proposed algorithmic framework is proposed, namely the construction of the probabilistic state graph representing the underlying stream. To this end, the vertices of the graph are required along with the edges between them and the respective transition probabilities. Vertex loops are allowed and in fact in certain scenarios expected for both mainstream and rare events. The former from their very definition cover considerable stream parts and the latter may indicate an irrecoverable situation.

Once clustering yields the set of actual events Q in the stream as shown in (17), it gets to serve as the graph vertex set. In this way, the bijection between the event set and the vertex set is guaranteed by construction, avoiding thus any matching issues.

$$Q \triangleq \{q_1, \ldots, q_n\} \tag{17}$$

The softmax score, which is extensively used as a Bayesian estimator of the true values of long observation vectors and, moreover, can be interpreted as an inherent distribution of that vector, shown in (18) is used to compute the state transition probabilities.

$$\mathrm{prob}\left\{q_i \rightarrow q_j\right\} \triangleq \frac{\exp\left(n_{i,j}\right)}{\sum_{u,v} \exp\left(n_{u,v}\right)} = \frac{\exp\left(n_{i,j}\right)}{\exp\left(n\right)} \tag{18}$$

In (18) $n_{i,j}$ is the number of transitions between any item of the i-th cluster to any of the j-th one. Therefore, cluster connectivity patterns are reflected in the stream graph. Notice that by construction the graph is not symmetric in the sense that the transition probability from q_i to q_j is in the general case different than that of the reverse transition. This is especially useful in the easy identification of black swan states by examining the ratio of (19).

$$\max\left[\frac{\mathrm{prob}\left\{q_i \rightarrow q_j\right\}}{\mathrm{prob}\left\{q_j \rightarrow q_i\right\}}\right] = \max\left[\exp\left(n_{i,j} - n_{j,i}\right)\right] \tag{19}$$

The representation explained above has the advantage of summarizing a potentially very long stream to a very compact form which nevertheless maintains the original information regarding event occurrences in an intuitive manner. Moreover, it allows the easy and efficient comparison of streams while avoiding massive computations. Third, state graphs focus on stream mechanics rather than on their populations, allowing therefore transfer learning methodologies. Thus, two streams of comparable parameters coming from different populations but having the same stochastic behavior in terms of event dynamics will result in two identical state graphs. A last key point is that a stream may contain more than one such events, being generated by mechanisms engineered to reflect real life including unexpected or exceptional events and even having their own flaws. This leads to the question of whether there is a connection between these rare events and whether there is an intrinsic connection between them, revealing thus the true cause of their occurrence. Consider for instance the case of very large object oriented software systems deployed as part of a critical infrastructure and operating in real time. Then, the reason for the appearance of an exception can be a previous one, revealing thus a hidden fatal code flaw which could be otherwise overlooked.

However, the proposed methodology requires knowledge of the steady state stream distribution, namely of the true distribution over the items being streamed. This can be obtained only by appropriate sketches of the stream or at least of large segments thereof which are of statistical importance. Additionally, said distribution may be evolving over time according to dynamics dictated by the underlying original data generation processes. This stochastic evolution may well range from changing distribution parameters to jumping to a different one or even cycling through distributions.

4 RESULTS

4.1 Overview

The proposed reservoir sampling approach was tested using the Python 3.9 programming language and the development tool PyCharm. The following packages are needed for setup: choice and numpy. The trials were done on the following hardware: CPU i9-10850k, 32GB RAM, Firecuda 530 NVMe disc, and Windows 11 operating system. We refer to samples obtained as r.

The results of uniform sampling with and without replacement are represented in section 4.2, while the results of weighted sampling with or without replacement are given in section 4.3.

In this section comparing two discrete distributions is necessary. The Kullback-Leibler divergence of equation (20) is an established probabilistic asymmetric measure of the loss incurred when a discrete distribution p substitutes a reference one q defined as:

$$\langle p \,||\, q \rangle \triangleq \sum_k p_k \log_b \left(\frac{p_k}{q_k} \right) \tag{20}$$

The logarithm base b determines the units the aforementioned loss is measured in, bits in this case. For the special case when the reference distribution is the uniform one, the Kullback-Leibler divergence takes the special form of (21), which is the difference between the entropy of the uniform distribution u and that of p.

$$\langle p \,||\, u \rangle \triangleq \sum_{k=1}^{n} p_k \log_b (n p_k) = \log_b n + \sum_{k=1} p_k \log_b p_k \tag{21}$$

4.2 Uniform Based Sampling

We first create a continuous tensor x of size 10. For $r = 3$ of number of samples we perform:

(1) Sampling with Replacement
(2) Sampling without Replacement

The tensor obtained by scenario (1) is as follows:

tensor([3, 3, 9])
tensor([3, 9, 3])
tensor([3, 9, 3])
tensor([0, 3, 6])
tensor([9, 3, 6])

While for scenario (2) we obtain:

tensor([0, 3, 6])
tensor([0, 3, 6])
tensor([0, 3, 6])
tensor([0, 3, 9])
tensor([0, 3, 9])

We therefore define a tensor of size 10 on CUDA/GPU and it should be a contiguous tensor as shown in result below. Again for scenario (1) and (2) we obtain:

tensor([0, 0, 9], device='cuda:0')
tensor([3, 3, 9], device='cuda:0')
tensor([3, 6, 6], device='cuda:0')
tensor([0, 3, 9], device='cuda:0')
tensor([3, 0, 9], device='cuda:0')

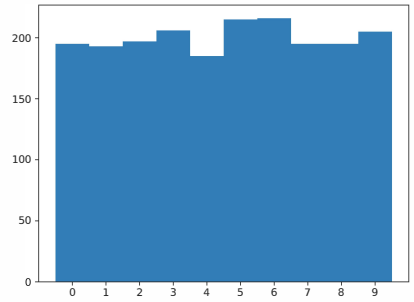
and

tensor([0, 3, 6], device='cuda:0')
tensor([9, 3, 6], device='cuda:0')
tensor([0, 3, 9], device='cuda:0')
tensor([0, 9, 6], device='cuda:0')
tensor([0, 3, 9], device='cuda:0')

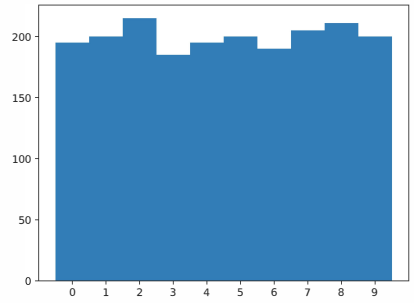
While we evaluate the sampling method against two libraries, *torch* and *numpy*. The evaluation metric for time is the average time per loop ± mean std.

The two assessment tools used for development and outcomes are CUDA, a parallel computing platform powered by NVIDIA, and CPP, a C++ API. The reservoir sampling code was built in C++, and the benchmarks in Python. Figures were created using matplotlib.

For the CPP run, the findings for $r = 2$ are presented in 1. We have a uniform distribution with substitution on 1a, while we have a uniform distribution without substitution on 1b.



(a) Uniform Based Sampling with Substitution



(b) Uniform Based Sampling without Substitution

Figure 1: Uniform Based Sampling with r=2 on CPP

The findings for $r = 8$ are presented in 2. We have a uniform distribution with replacement on 2a, while we have a uniform distribution without replacement on 2b.

For the CUDA run, the findings for $r = 2$ are presented in 3. We have a uniform distribution with replacement on 3a, while a uniform distribution without replacement is on 3b.

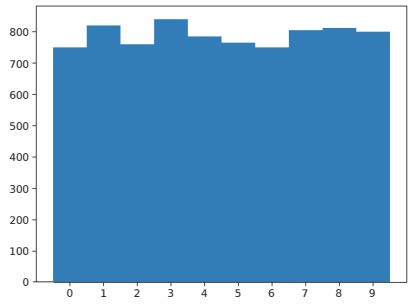
The findings for r=8 are presented in 4. We have a uniform distribution on 4a, while a uniform distribution without replacement is in 4b.

The results of evaluating the uniform distribution without using a substitute for $r = 100$ are displayed in table 2.

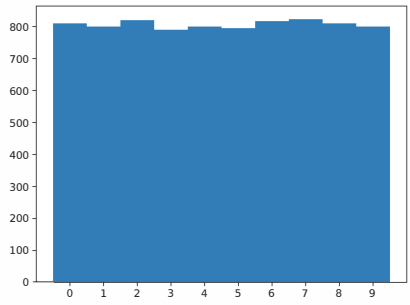
Table 2: Evaluation of Uniform Sampling for r=100.

r	Method	Substitution	Time Range
100	Torch	True	5.72 μs ± 15.89 ns
100	Numpy	True	17.4 μs ± 73.21 ns
100	Torch	False	14.3 μs ± 101 ns
100	Numpy	False	129 μs ± 2.08 μs

The results of evaluating the uniform distribution without using a substitute for $r = 4500$ are displayed in table 3.

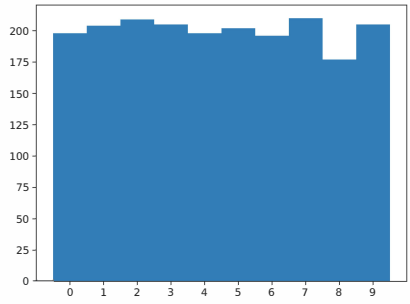


(a) Uniform Based Sampling with Substitution

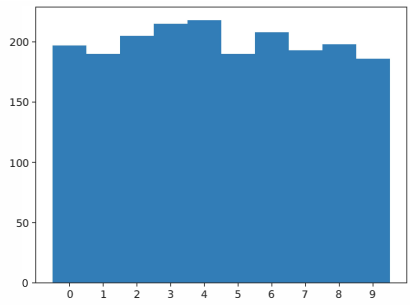


(b) Uniform Based Sampling without Substitution

Figure 2: Uniform Based Sampling with r=8 on CPP

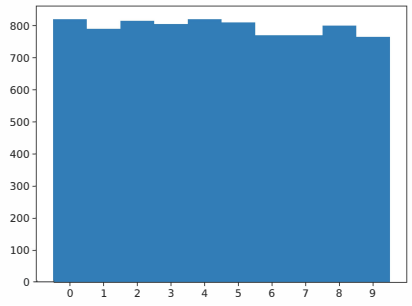


(a) Uniform Based Sampling with Substitution

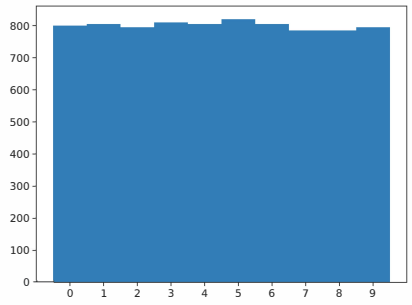


(b) Uniform Based Sampling without Substitution

Figure 3: Uniform Based Sampling r=2 on CUDA



(a) Uniform Based Sampling with Substitution



(b) Uniform Based Sampling without Substitution

Figure 4: Uniform Based Sampling with r=8 on CUDA

Table 3: Evaluation of Uniform Sampling for r=4500.

r	Method	Substitution	Time Range
4500	Torch	True	53.21 μs ± 1.43 μs
4500	Numpy	True	71.92 μs ± 152 ns
4500	Torch	False	72.57 μs ± 93.59 ns
4500	Numpy	False	71.71μs ± 257 ns

The results of evaluating the uniform distribution without using a substitute for $r = 9000$ are displayed in table 4.

Table 4: Evaluation of Uniform Sampling for r=9000.

r	Method	Substitution	Time Range
9000	Torch	True	95.41 μs ± 488 ns
9000	Numpy	True	125 μs ± 1.15 μs
9000	Torch	False	48.42 μs ± 674 ns
9000	Numpy	False	137 μs ± 1.41 μs

4.3 Weight Based Sampling

For the CPP run the results of assessing weighted sampling without substitution for $r = 100$ are summarised in table 5.

The results of assessing weighted sampling without substitution for $r = 4500$ are summarised in table 6.

Table 5: Evaluation of Weighted Sampling for r=100.

r	Method	Substitution	Time Range
100	Torch	True	43.7 μs ± 536 ns
100	Numpy	True	110 μs ± 903 ns
100	Torch	False	235 μs ± 3.67 ns
100	Numpy	False	172 μs ± 2.3 μs

Table 6: Evaluation of Weighted Sampling for r=4500.

r	Method	Substitution	Time Range
4500	Torch	True	407 μs ± 4.7 μs
4500	Numpy	True	495 μs ± 8.65 μs
4500	Torch	False	295 μs ± 600 ns
4500	Numpy	False	1.27 ms ± 4.14 μs

The results of assessing weighted sampling without substitution for $r = 9000$ are summarised in table 7.

Table 7: Evaluation of Weighted Sampling for r=9000.

r	Method	Substitution	Time Range
9000	Torch	True	773 μs ± 1.3 μs
9000	Numpy	True	872 μs ± 1.14 μs
9000	Torch	False	373 μs ± 999 ns
9000	Numpy	False	2.92 ms ± 6.24 μs

The findings for r=1 are shown in 5. We have weighted sampling with replacement in 5a, while we have weighted sampling without replacement in 5b.

For the CUDA run The findings for $r = 1$ are shown in 6. We have weighted sampling with replacement in 6a, while we have weighted sampling without replacement in 6b.

In case of weighted sampling, the time take is directly proportional to number of samples to take. So for k=9000, time is highest and for $k = 100$, time is lowest as shown in table 5.

4.4 Performance for multi-d tensors

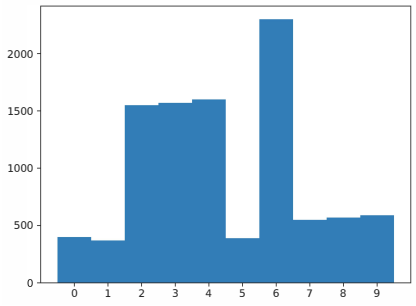
By creating a two-dimensional tensor of 10000 elements we run the experiments for (3) scenarios.

(1) r is small ⟶ r = 100
(2) r is medium ⟶ r = 4500
(3) r is large ⟶ r = 9000

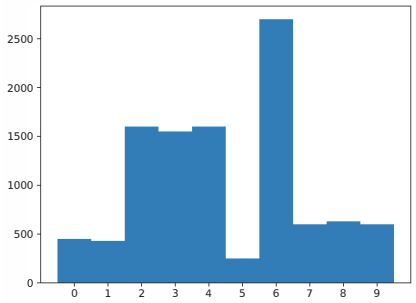
Then we evaluate the running times. The results are shown in table 8.

As shown in the preceding table, the time to perform sampling on tensor(x) was low for small number of samples with elemental substitution while for $r = 9000$ the best time achieved was without substitution.

The overall results for the proposed sampling technique are shown in table 10. The best sampling times are depicted in bold. We can observe that the best results come from torch tensors while they outperform numpy in both uniform and weighted methods

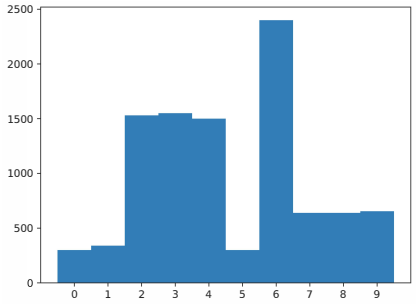


(a) Weight Based Sampling with Substitution

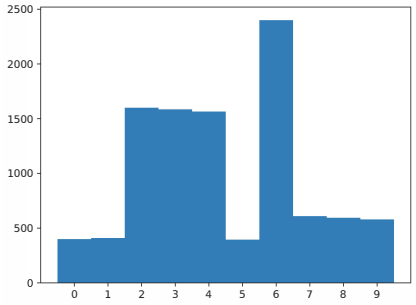


(b) Weight Based Sampling without Substitution

Figure 5: Weight Based Sampling with r=1 on CPP



(a) Weight Based Sampling with Substitution



(b) Weight Based Sampling without Substitution

Figure 6: Weight Based Sampling with r=1 on CUDA

Table 8: Performance for multi-d tensors.

r	Method	Substitution	Time Range
100	Tensor(x)	True	5.79 μs ± 5.46 ns
100	Tensor(x)	False	14.3 μs ± 16.1 ns
100	2D-Tensor(x)	None	130 μs ± 209 ns
4500	Tensor(x)	True	52.5 μs ± 32.5 ns
4500	Tensor(x)	False	72.9 μs ± 681 ns
4500	2D-Tensor(x)	None	151 μs ± 389 ns
9000	Tensor(x)	True	95.4 μs ± 75.5 ns
9000	Tensor(x)	False	45.8 μs ± 56.6 ns
9000	2D-Tensor(x)	None	167 μs ± 3.14 μs

Table 9: Evaluation of Weighted Sampling for r=9000.

r	Method	Substitution	Time Range
100	Torch	True	5.79 μs ± 5.46 ns
100	Numpy	True	872 μs ± 1.14 μs
100	Torch	False	373 μs ± 999 ns
100	Numpy	False	2.92 ms ± 6.24 μs

except for r=4500 where numpy outperformed torch for about one microsecond.

Table 10: Overall comparison across all methods.

Method	Type	Size	Samples	Replacement	Time
Numpy	Uniform	10^5	9×10^3	True	126 μs
Numpy	Weighted	10^5	9×10^3	True	872 μs
Torch	Tensor	10^5	9×10^3	True	**95.4** μs
Numpy	Uniform	10^5	1×10^2	True	17.3 μs
Numpy	Weighted	10^5	1×10^2	True	110 μs
Torch	Tensor	10^5	1×10^2	True	**5.79** μs
Numpy	Uniform	10^5	45×10^2	True	71.9 μs
Numpy	Weighted	10^5	45×10^2	True	495 μs
Torch	Tensor	10^5	45×10^2	True	**52.7** μs
Numpy	Uniform	10^5	9×10^3	False	138 μs
Numpy	Weighted	10^5	9×10^3	False	2920 μs
Torch	Tensor	10^5	9×10^3	False	**45.8** μs
Numpy	Uniform	10^5	1×10^2	False	130 μs
Numpy	Weighted	10^5	1×10^2	False	172 μs
Torch	Tensor	10^5	1×10^2	False	**14.3** μs
Numpy	Uniform	10^5	45×10^2	False	**71.7** μs
Numpy	Weighted	10^5	45×10^2	False	1270 μs
Torch	Tensor	10^5	45×10^2	False	72.9 μs

4.5 Proposed Method for Clustering

For training and testing, the suggested alternative of k-means was utilized to detect events from IoT sensors using the SENSORS data set. The suggested res-means algorithm detects events as indicated in the table 13.

Table 11: Overall results of tensors across two different tools.

Tensor	Tool	Size	Samples	Replacement	Time
Uniform	CPU	10^7	10^5	True	208 μs
Uniform	CUDA	10^7	10^5	True	27.1 μs
Uniform	CPU	10^7	10^5	False	6.87 μs
Uniform	CUDA	10^7	10^5	False	3.33 μs
Weighted	CPU	10^7	10^5	False	335 μs
Weighted	CUDA	10^7	10^5	False	84 μs

4.5.1 Evaluation. Three distinct methodologies were examined to evaluate the suggested k-meas alternative. DBSCAN and E-DBSCAN are the typical k-means. The findings are tabulated in 12. The evaluation metrics used in this work are precision, recall, and F-score.

Table 12: Performance evaluation of the proposed method.

Metrics	K-means	DBSCAN	E-DBSCAN	Res-Means
Precision	39.7	49.1	53.23	51.2
Recall	25.7	40.8	50.1	44.8
F-Score	30.46	44.64	51.45	47.7

4.5.2 Event Detection. The table 13 lists the detected events. The event-detection frameworks CM, BoW, and EBoW are all quite similar. The data set was classified into several categories, and the suggested approach clustered all of the characteristics. Following that, the four characteristics are shown as four clusters with centroids and outliers. The events shown here are the total of the outliers divided by the cluster center's distance. Once again, our technique surpasses the other three in each of the four areas.

Table 13: Events detected with the proposed method.

Event Detected	CM	BoW	EBoW	Res-means
Faulty sensor	4.81	7.12	7.91	**8.44**
Noisy data	9.23	9.8	11.29	**12.71**
Asynchronous intervals	4.38	5.23	7.8	**9.47**
Communication intangibility	5.62	6.45	8.4	**11.25**

5 CONCLUSIONS AND FUTURE WORK

In the context of this work, the inner workings of weighted reservoir sampling are presented along with an algorithmic framework for constructing probabilistic state graphs from events discovered in evolving streams consisting of three distinct steps. First, weighted reservoir sampling identifies important values in the stream. Then, they are clustered either with k-means or DBSCAN so that regular values are grouped together in mainstream events, whereas outliers are considered as black swan events. The procedure is further made simpler by supplying just a small number of data bits that are indicative of the data set being used. Finally, a probabilistic state

graph is constructed based on these events so that streams can be efficiently represented and compared between them.

Improving reservoir sampling accuracy and precision while keeping complexity within reasonable levels is a long range research goal. Future enhancements may involve sophisticated clustering or even consensus clustering taking place in the second step. The ability to compute the two algorithms at the same time might be a significant gain in overall performance and efficiency. Also worthy of mention is the extension of the proposed framework so that it can track changes in the distribution of the data. Ultimately, lightweight implementations as the one presented can adapt to IoT scenarios.

REFERENCES

[1] Marcel R Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. 2012. Streamkm++ a clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)* 17 (2012), 2–1.

[2] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. 2003. A Framework for Clustering Evolving Data Streams. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29 (VLDB '03)*. VLDB Endowment, Berlin, Germany, 81–92.

[3] B Angelin and A Geetha. 2020. Outlier Detection using Clustering Techniques–K-means and K-median. In *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, 373–378.

[4] David Arthur and Sergei Vassilvitskii. 2006. *k-means++: The advantages of careful seeding*. Technical Report. Stanford.

[5] Azzedine Boukerche, Lining Zheng, and Omar Alfandi. 2020. Outlier detection: Methods, models, and classification. *ACM Computing Surveys (CSUR)* 53, 3 (2020), 1–37.

[6] Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. 2009. Optimal sampling from sliding windows. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 147–156.

[7] Pierre Brémaud. 2013. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*. Vol. 31. Springer Science & Business Media.

[8] Toon Calders, Nele Dexters, Joris JM Gillis, and Bart Goethals. 2014. Mining frequent itemsets in a stream. *Information Systems* 39 (2014), 233–255.

[9] David Camacho, Ma Victoria Luzón, and Erik Cambria. 2021. New research methods and algorithms in social network analysis. , 290–293 pages.

[10] Feng Cao, Martin Estert, Weining Qian, and Aoying Zhou. 2006. Density-based clustering over an evolving data stream with noise. In *Proceedings of the Sixth SIAM International Conference on Data Mining*. SIAM, SIAM, Bethesda, MD, USA, 328–339.

[11] Jian Chen and Jeffrey S Rosenthal. 2012. Decrypting classical cipher text using Markov chain Monte Carlo. *Statistics and Computing* 22, 2 (2012), 397–413. https://doi.org/10.1007/s11222-011-9232-5

[12] Ting Chen, Yizhou Sun, Yue Shi, and Liangjie Hong. 2017. On sampling strategies for neural network-based collaborative filtering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 767–776.

[13] Yun Chi, Haixun Wang, P.S. Yu, and R.R. Muntz. 2004. Moment: maintaining closed frequent itemsets over a stream sliding window. In *Fourth IEEE International Conference on Data Mining (ICDM'04)*. IEEE, Brighton, UK, 59–66. https://doi.org/10.1109/ICDM.2004.10084

[14] Mark Ming-Tso Chiang and Boris Mirkin. 2010. Intelligent choice of the number of clusters in k-means clustering: an experimental study with different cluster spreads. *Journal of classification* 27, 1 (2010), 3–40.

[15] Ariel Debrouvier et al. 2021. A model and query language for temporal graph databases. *VLDB Journal* 30, 5 (2021), 825–858.

[16] Wei Deng, Guang Lin, and Faming Liang. 2022. An adaptively weighted stochastic gradient MCMC algorithm for Monte Carlo simulation and global optimization. *Statistics and Computing* 32, 4 (2022), 1–24.

[17] Pavlos S Efraimidis and Paul G Spirakis. 2006. Weighted random sampling with a reservoir. *Information processing letters* 97, 5 (2006), 181–185.

[18] Ahmed Eldawy, Yuan Li, Mohamed F. Mokbel, and Ravi Janardan. 2013. CG_Hadoop: Computational Geometry in MapReduce. In *SIGSPATIAL*. ACM, NY, USA, 294–303. https://doi.org/10.1145/2525314.2525349

[19] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. 2000. Clustering data streams. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*. IEEE, Redondo Beach, CA, USA, 359–366. https://doi.org/10.1109/SFCS.2000.892124

[20] John A. Hartigan. 1975. *Clustering Algorithms* (99th ed.). John Wiley & Sons, Inc., USA.

[21] Saeid Iranmanesh et al. 2019. Novel DTN mobility-driven routing in autonomous drone logistics networks. *IEEE Access* 8 (2019), 13661–13673.

[22] Ruoming Jin and Gagan Agrawal. 2007. *Frequent Pattern Mining in Data Streams*. Springer US, Boston, MA, 61–84. https://doi.org/10.1007/978-0-387-47534-9_4

[23] Aristeidis Karras and Christos Karras. 2022. Integrating User and Item Reviews in Deep Cooperative Neural Networks for Movie Recommendation. *arXiv preprint arXiv:2205.06296* (2022).

[24] Christos Karras and Aristeidis Karras. 2022. DBSOP: An Efficient Heuristic for Speedy MCMC Sampling on Polytopes. *arXiv preprint arXiv:2203.10916* (2022).

[25] Christos Karras, Aristeidis Karras, Markos Avlonitis, Ioanna Giannoukou, and Spyros Sioutas. 2022. Maximum Likelihood Estimators on MCMC Sampling Algorithms for Decision Making. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, Cham, 345–356.

[26] Christos Karras, Aristeidis Karras, Markos Avlonitis, and Spyros Sioutas. 2022. An Overview of MCMC Methods: From Theory to Applications. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, Cham, 319–332.

[27] Christos Karras, Aristeidis Karras, and Spyros Sioutas. 2022. Pattern Recognition and Event Detection on IoT Data-streams. *arXiv preprint arXiv:2203.01114* (2022).

[28] George Lagogiannis, Stavros Kontopoulos, and Christos Makris. 2019. On the randomness that generates biased samples: The limited randomness approach. *Computer Science and Information Systems* 16, 1 (2019), 205–225.

[29] Aihua Li, Weijia Xu, Zhidong Liu, and Yong Shi. 2021. Improved incremental local outlier detection for data streams based on the landmark window model. *Knowledge and Information Systems* 63, 8 (2021), 2129–2155.

[30] Yuan Li, Ahmed Eldawy, Jie Xue, Nadezda Knorozova, Mohamed F Mokbel, and Ravi Janardan. 2019. Scalable computational geometry in MapReduce. *VLDB Journal* 28, 4 (2019), 523–548.

[31] Fernando Llorente, E Curbelo, Luca Martino, Victor Elvira, and David Delgado. 2022. MCMC-driven importance samplers. *Applied Mathematical Modelling* (2022).

[32] David Luengo, Luca Martino, Mónica Bugallo, Víctor Elvira, and Simo Särkkä. 2020. A survey of Monte Carlo methods for parameter estimation. *EURASIP Journal on Advances in Signal Processing* 2020, 1 (2020), 1–62.

[33] Mohammad Sultan Mahmud, Joshua Zhexue Huang, Salman Salloum, Tamer Z Emara, and Kuanishbay Sadatdiynov. 2020. A survey of data partitioning and sampling methods to support big data analysis. *Big Data Mining and Analytics* 3, 2 (2020), 85–101.

[34] Tomas Martin, Guy Francoeur, and Petko Valtchev. 2020. CICLAD: A Fast and Memory-Efficient Closed Itemset Miner for Streams. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 1810–1818. https://doi.org/10.1145/3394486.3403232

[35] Luca Martino, Víctor Elvira, David Luengo, Jukka Corander, and Francisco Louzada. 2016. Orthogonal parallel MCMC methods for sampling and optimization. *Digital Signal Processing* 58 (2016), 64–84.

[36] L. Martino, V. Elvira, D. Luengo, J. Corander, and F. Louzada. 2016. Orthogonal parallel MCMC methods for sampling and optimization. *Digital Signal Processing* 58 (2016), 64–84. https://doi.org/10.1016/j.dsp.2016.07.013

[37] Eli Packer, Peter Bak, Mikko Nikkilä, Valentin Polishchuk, and Harold J. Ship. 2013. Visual analytics for spatial clustering: Using a heuristic approach for guided exploration. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2179–2188. https://doi.org/10.1109/TVCG.2013.224

[38] Marcelo Pereyra, Philip Schniter, Emilie Chouzenoux, Jean-Christophe Pesquet, Jean-Yves Tourneret, Alfred O Hero, and Steve McLaughlin. 2015. A survey of stochastic simulation and optimization methods in signal processing. *IEEE Journal of Selected Topics in Signal Processing* 10, 2 (2015), 224–241.

[39] Chedy Raissi and Pascal Poncelet. 2007. Sampling for Sequential Pattern Mining: From Static Databases to Data Streams. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, Omaha, NE, USA, 631–636. https://doi.org/10.1109/ICDM.2007.82

[40] Pang-Ning Tan, Michael S. Steinbach, and Vipin Kumar. 2022. Introduction to Data Mining. *Data Mining and Machine Learning Applications* (2022).

[41] Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong, and Young-Koo Lee. 2009. Sliding window-based frequent pattern mining over data streams. *Information sciences* 179, 22 (2009), 3843–3865.

[42] Hongzhi Wang, Mohamed Jaward Bah, and Mohamed Hammad. 2019. Progress in outlier detection techniques: A survey. *Ieee Access* 7 (2019), 107964–108000.

[43] Tongxin Wang et al. 2021. MOGONET integrates multi-omics data using graph convolutional networks allowing patient classification and biomarker identification. *Nature Communications* 12, 1 (2021), 1–13.

[44] Ming-Juan Wu et al. 2019. Integrative hypergraph regularization principal component analysis for sample clustering and co-expression genes network analysis on multi-omics data. *IEEE Journal of Biomedical and Health Informatics* 24, 6 (2019), 1823–1834.

[45] Linli Xu and Dale Schuurmans. 2005. Unsupervised and Semi-Supervised Multi-Class Support Vector Machines. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2 (AAAI'05)*. AAAI Press, Pittsburgh, Pennsylvania, 904–910.