NEW QUERY AND ANALYTICS OVER LARGE SEQUENCE DATA

– A STUDY ON STREAKS AND STREAM

RAN BAI

PhD

The Hong Kong Polytechnic University

2019

The Hong Kong Polytechnic University

Department of Computing

New Query and Analytics over Large Sequence Data – a Study on Streaks and Stream

Ran BAI

A thesis submitted in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

April 2019

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

.....

Ran BAI

April 2019

Abstract

Sequence data consists of ordered items or elements. Query processing and analytics on large sequence data have many research challenges. This thesis studies two important problems in this domain.

The first part of this thesis studies a new problem of finding "historic moments" from sequence data. Specifically, we introduce a new concept called "historic moments", which is motivated from real applications such as computational journalism. We present algorithms to efficiently compute historic moments from sequence data. The algorithm is *incremental* and *space-optimal*, meaning that when facing new data arrival, it is able to efficiently refresh the results by keeping minimal information. Case studies show that historic moments can significantly improve the insights offered by prominent streaks alone.

The second part of this thesis studies another new problem of answering range-count query over data stream. Specifically, in applications such as network monitoring, telecommunication analysis, and sensor measurements, massive amounts of data arrive as a high-rate stream and real-time analytic over the stream data is required. Maintaining a succinct synopsis structure called sketch over the data stream has been a dominant approach to support analysis in those applications. Recent applications, however, demand more sophisticated types of queries and range-count query is our focus in this thesis. Unfortunately, state-of-the-art sketches perform poorly when facing range-counting as none of them was designed to support range-count queries at the outset. In this thesis,

we aim to fill the gap and present LSH-Sketch, a sketch that supports range-counting over rapid data stream. As a sketch that supports range-count queries, LSH-Sketch can naturally support point-count queries as well. As its name suggests, LSH-Sketch is based on the use of locality sensitive hashing. Like the classic CM-Sketch, LSH-Sketch is also a core sketch that many sketch variants and applications can be built on top. Empirical results show that LSH-Sketch's insertion throughput is as good as CM-Sketch and it outperforms CM-Sketch in terms of accuracy and query throughput under all query ranges. LSH-Sketch thus has the potential to replace CM-Sketch to serve as the core sketch in multiple application domains.

Publications arising from the thesis

Ran Bai, Wing Kai Hon, Eric Lo, Zhian He, and Kenny Zhu. 2019. Historic Moments Discovery in Sequence Data. ACM Trans. Database Syst. 44, 1, Article 3 (January 2019), 33 pages. DOI: https://doi.org/10.1145/3276975

Acknowledgements

I would first like to thank my supervisors Dr. Eric LO, Dr. Ken YIU for providing much advice and help on my research in the past four years. I would also like to thank Dr. Wing Kai HON for discussing and giving comments on my thesis.

I would like to thank my workmates from DB group, Dr. Zhian HE, Dr. Yu LI, Dr. Bo TANG, Dr. Petrie WONG, Dr. Qiang ZHANG, Dr. Wenjian XU, Ziqiang FENG, Chris LIU, Pengfei ZHANG, Xuxuan ZHOU and Baotong LU. I would also like to thank my mates in our office, Dr. Feng TAN, Dr. Shuhang GU, Dr. Yanxing HU, Dr. Shang GAO, Dr. Zhe PENG, Dr. Yi DOU.

I would also like to thank my friends for supporting me. I would like to thank Qian WANG, Peiran YU, Xingyu JING, Jalynna Jiajing LIANG, Yu LIU, Dr. Shuang DONG, Dr. Si CHEN, Yingqiao YANG and Supitcha Jutatungcharoen. I would also like to thank my teacher Suki YIP.

Finally, I must express my profound gratitude to my parents for their love.

Table of contents

Declaration **i**

Abstract **iii**

Table of contents **vii**

List of Figures **xi**

List of Tables **xv**

1 Introduction **1**

1.1 Historic Moments Discovery in Sequence Data 2

1.2 Range Counting Over Data Stream 5

2 Historic Moments Discovery in Sequence Data **9**

2.1 Related Work 9

2.2 Problem Definition 13

2.3 Finding Historic Moments from a Data Sequence 18
2.3.1 Baseline Algorithm (BA) 19
2.3.2 Baseline Incremental Algorithm (BIA) 21
2.3.3 Space Optimal Incremental Algorithm (SOIA) 25
2.4 Case Study 49
2.4.1 Microsoft's stock price 49
2.4.2 Beijing's temperature 50
2.4.3 Taiwan seismic datasets 51
2.5 Performance Study 53
2.5.1 Overall Comparison 54
2.5.2 Historic Moment Exploration with Data Update 56
2.5.3 Sensitivity Study 59
2.6 Summary 61

3 Range Counting over Data Stream 63

3.1 Preliminary and Background 63
3.1.1 Problem Definition 64
3.1.2 Sketch 64
3.1.3 Locality Sensitive Hashing (LSH) 69
3.2 LSH-Sketch 70

3.2.1 Insertion 73

3.2.2 Range-Counting: Algorithm AOC 73

3.2.3 Accuracy 75

3.2.4 Implementation of (r_1, r_2, p_1, p_2)-sensitive family for LSH-Sketch 78

3.3 Experiment 82

3.3.1 Accuracy 84

3.3.2 Insertion Throughput 88

3.3.3 Query Throughput 92

3.3.4 Accuracy and Space Trade-off 92

3.4 Related Work 93

3.5 Summary 98

4 Conclusion and Future Work **99**

Appendices **103**

A Appendix **103**

A.0.1 BIA-MS 105

A.0.2 SOIA-MS 105

A.0.3 Performance Study 107

Bibliography **113**

List of Figures

1.1 Thesis Scope 2

2.1 A data sequence (a value is represented by $\times$) 10

2.2 Data Sequence D_{19} 14

2.3 After v_{20} is appended 25

2.4 BIA vs. SOIA 55

2.5 BIA vs. SOIA under data update (D1, D2, D3) 57

2.6 BIA vs. SOIA under data update (D4, D5) 58

2.7 Varying σ 59

2.8 Varying σ When Updating 60

2.9 Varying k 60

3.1 CM-Sketch Structure 66

3.2 Ability to adjust over-counting 71

3.3 LSH-Sketch using AOC to do range counting 74

3.4 Range Count Accuracy (log-log scale) on Kosorak and WebDocs datasets 85

3.5 Range Count Accuracy (log-log scale) on CAIDA and Zip-0 datasets 86

3.6 Range Count Accuracy (log-log scale) on Zip-1 and Zip-2 datasets 87

3.7 Insertion Throughput 88

3.8 Query Throughput 88

3.9 Vary the Width of LSH-Sketch (log-log scale) on Kosorak and WebDocs Datasets 89

3.10 Vary the Width of LSH-Sketch (log-log scale) on CAIDA and Zip-0 Datasets 90

3.11 Vary the Width of LSH-Sketch (log-log scale) on Zip-1 and Zip-2 Datasets 91

3.12 Vary the Depth of LSH-Sketch (log-log scale) on Kosorak and WebDocs Datasets 94

3.13 Vary the Depth of LSH-Sketch (log-log scale) on CAIDA and Zip-0 Datasets 95

3.14 Vary the Depth of LSH-Sketch (log-log scale) on Zip-1 and Zip-2 Datasets 96

A.1 SOIA-MS vs. BIA-MS 110

A.2 SOIA-MS vs BIA-MS under data update (MS1 MS2 MS3) 111

A.3 SOIA-MS vs BIA-MS under data update (MS4 MS5) 112

List of Tables

1.1 Sketches for range-counting under the same space budget (**bold** for this thesis's result) 6

2.1 Major Notations in Chapter 2 18

2.2 Summary of Datasets 54

2.3 Number of Streaks Maintained by BIA and SOIA 56

Chapter 1

Introduction

Sequence data consists of ordered items or elements. In our life, Sequences are a kind of data of great importance because it occur frequently in many fields such as finance, business, security and other applications, where the analysis of the sequence data needs to be executed in an efficient manner. Especially for "large" sequence data: query processing and analytics on large sequence data have many research challenges.

From two different aspects that cause "large" sequence data, this thesis studies new query and analytics respectively. Figure 1.1 shows the scope of the thesis.

For data sequence of large volume, this thesis studies how to find and discover interesting facts and streaks over them. We define a new concept called "historic moments" for sequence data discovery, present relevant algorithms to compute it efficiently, and prove that the facts found are insightful.

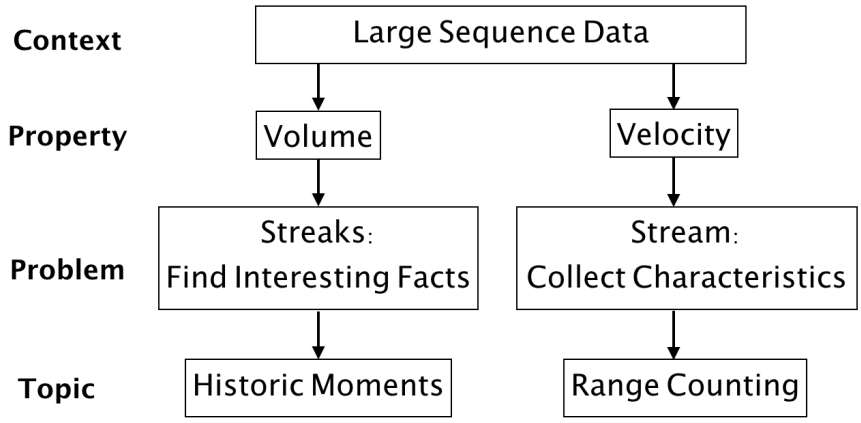


Figure 1.1: Thesis Scope

For data sequence of large velocity, collecting its characteristics poses great challenge, especially when the sequence is in the form of data stream and can be examined by single-pass only. This thesis studies a new problem of answering range-count query, i.e., a kind of statistics, over rapid data stream. We present LSH-Sketch with both good accuracy and efficiency.

1.1 Historic Moments Discovery in Sequence Data

Finding *prominent streaks* [33], a set of maximal contiguous subsequences with values all above (or below) a certain threshold, from a sequence dataset, has recently found applications in social network analysis, disease outbreak detection, and computational journalism [15, 30]. Take computational journalism as an example, the following news article in January 2011[1] contains a real example of

[1]Excerpts from V. Wagner's LiveFromBeijing blog:
`http://www.livefrombeijing.com/2011/01/beijing-breaks-record-for-longest-streak-of-consecutive-blue-sky-days-best-air-quality-in-years/`

prominent streak:

> *"Today is Beijing's 36th consecutive Blue Sky Day, a day whose Air Pollution Index (API) is 100 or below, indicating "excellent" or "good" air quality. As far as I can tell, this is the longest consecutive streak of Blue Sky Days in Beijing for at least ten years."*

In the excerpt above, the prominent streak refers to a *subsequence of 36 days consecutive measures of air pollution index of 100 or below*, and the authors in [33] have developed an efficient algorithm to discover streaks like this. Although useful, a prominent streak only stands for a singular event in the dataset. In fact, the news excerpt above is continued like this:

> *" ... in Beijing for at least ten years.* ***Previously, there were only three streaks of 30 days or longer, one in 2006 and two during 2008 Olympics****. "*

The last sentence is crucial: it pinpoints the *rarity* of the 36-day Blue Sky streak, otherwise readers who are unfamiliar with Beijing's weather would probably find the news mundane. In contrast, when it is further explained that the last time Beijing had 30 or more consecutive Blue Sky Days was nearly three years ago, readers will then be impressed by how rare the current streak is, and may be aroused to learn more about Beijing's environment. In other words, the three prominent streaks in 2006 and 2008 are similar "historic moments" happened before, which highlights the rarity of 36-day streak happened in 2011.

In this thesis, we formally introduce the concept of *historic moment* of a

streak s, which is a set of prominent streaks that end before s and can be used to highlight the *interestingness* of s. The term interestingness can take in many forms, but mainly centered around whether a *similar* event happened before, and if so, *how long ago it was*. The technical concern is how to efficiently report historic moments from a sequence dataset, with a consideration that the data sequences are being appended regularly (e.g., hourly update of crude oil price[2], update the seismic magnitude per one-tenth second[3]). To this end, we present a highly efficient incremental algorithm that can enable interactive historic moment analysis on a sequence dataset with continuous data update. The efficiency of the algorithm comes from maintaining an index in *minimal space*. Space-optimality leads to disk I/O reduction per operation or even makes the index small enough to be memory-resident. That property is crucial for online analysis and real-time monitoring, especially when there are possibly many data sequences of interest concurrently. Experiments on five real datasets show that our algorithm, namely space optimal incremental algorithm (SOIA), outperforms the baseline algorithm by 9× to 184× in terms of speed, using 98% to 99.5% less space (e.g., in our case study, our algorithm maintained an index of only 2GB for a 350GB data sequence whereas the baseline needed to maintain an index of size 500GB). Furthermore, our case studies show that historic moments indeed can help journalists to find full news stories, instead of half-baked ones found by prominent streaks, and help seismologists to get a bigger picture when analyzing ground motion data.

[2] `http://www.pmbull.com/oil-price/`
[3] `http://ds.iris.edu/ds/nodes/dmc/data/`

1.2 Range Counting Over Data Stream

Numerous data-intensive applications require query processing over data streams. Examples of such applications include network monitoring [65, 67], sensor networks [56, 61], and high-frequency financial trading [1, 22], to name a few. Recently, an emerging number of streaming applications [4, 13, 18, 24, 54, 62] are demanding richer forms of query types in addition to basic queries such as *point-count* queries [47, 53] and *heavy-hitter* queries [7, 8]. For example, network monitoring has been heavily relied on those two types of queries to monitor the network traffic. In there, a "flow" is a tuple that consists of the information about the IP addresses, ports, and the communication protocol between a source and a destination [40]. When monitoring the network, a point-count query specifies a value of an attribute as the "flowkey" (e.g., the source IP address) and counts the number of flows (tuples) that match that flowkey so far [7]. In their context, such a point-count query is called "flow size estimation". Lately, network monitoring has been advanced from basic queries to more advanced queries like *range-count* queries [31]. Instead of specifying a single value as the flowkey, a range of values (e.g., from `178.115.5.1` to `178.115.263.999`) is specified instead and its target is to get the aggregated number of flows (tuples) in that range. Range-count queries enable more sophisticated kinds of analysis called "hyperflow size estimation", which in turn offers better network management and security [19]. Wireless sensor network [35, 36, 43] is another application that demands range-count queries. Like network devices, sensors also have very limited storage [28]. Range-count queries are prevalent there because there are queries that count the number of sensors whose values are in a certain range.

Query processing over data streams is challenging because of the high incoming throughput of data (e.g., up to 100Gbps in network monitoring [17]), which rules out the basic choice of storing all incoming data and leaves behind the only option of having a "single-pass" over the data [3]. To support count queries over rapid data streams, one promising direction is to maintain a *synopsis structure* [16,23,51,52,66] over the incoming data stream and use that to answer various queries. Synopsis structures typically trade the answer quality to meet the requirements of (i) high incoming throughput, (ii) low query latency, and (iii) limited working space. Example synopsis structures for queries over stream include all kind of "sketches" (e.g., CM-Sketch [16], CU-Sketch [23], and CML-Sketch [51]). Generally, sketches can return *approximate* answer with *theoretical guarantees*.

For Range-Counting	Insertion Latency	Query Latency	Absolute Error
Point-Count Sketches [16, 20, 23, 51]	$O(1)$	$\boldsymbol{O(\lvert Q\rvert)}$	$\boldsymbol{\varepsilon n\lvert Q\rvert}$
Multi-Level Sketches [16]	$O(\log D)$	$O(\log \lvert Q\rvert)$	$2\varepsilon n \log D \log \lvert Q\rvert$
LSH-Sketch	$\boldsymbol{O(1)}$	$\boldsymbol{O(\lvert Q\rvert\sqrt{w/D})}$	$\boldsymbol{(4\varepsilon + (\sqrt{2\varepsilon \mathrm{e} D} + \mathrm{e})\frac{\lvert Q\rvert}{D})n}$

Table 1.1: Sketches for range-counting under the same space budget (**bold** for this thesis's result)

Although many sketches exist, to our best knowledge, none of them has ever been designed for range-count queries in the first place. Given the increasing importance of range-count queries in data stream applications, our work aims to fill the gap. Currently, to answer range-count queries over data stream, we can either (i) use point-count sketches such as CM-sketch [20] or (ii) their multi-level extension [16]. Table 1.1 summarizes the state-of-the-art techniques that can support approximate range-counting on data streams. Point-count sketches can answer range-count queries in an awkward sense and the error is large: $\varepsilon n|Q|$,

where n is the number of items appeared in the stream so far, $|Q|$ is query range size, and ε is a parameter derived from the space budget. Multi-level extension of point-count sketches can answer range-count queries with a smaller error: $2\varepsilon n \log D \log |Q|$, where D is size of domain of the attribute-of-interest. However, its reduced error comes with a cost—its insertion latency degrades from $O(1)$ to $O(\log D)$, which is unacceptable in modern data stream applications when this translates to insertion throughput.

In this thesis, we present LSH-sketch, a sketch designed for range-count queries over rapid data stream. As a synopsis structure for approximate range-count queries, LSH-sketch naturally can support point-count queries as well (which is just a special case of a range of size 1). Furthermore, LSH-sketch preserves the excellent $O(1)$ insertion latency like point-count sketches and has an error of $(4\varepsilon + (\sqrt{2\varepsilon \mathrm{e} D} + \mathrm{e})\frac{|Q|}{D})n$, which is generally better than the error of point-count sketches. The name of LSH-sketch hints the technique behind its excellence—the use of *locality sensitive hashing* (LSH). Specifically, when it comes to range queries, it is known that *histograms* like V-optimal histogram [29, 32] provide very good error guarantees. However, histograms can never gain a foothold in stream query processing because their theoretical guarantees require building the histogram by sorting or scanning the data multiple times [12, 32]. Interestingly, like all existing sketches, the use of locality sensitive hashing (LSH) does not require sorting or multiple passes on the data, while at the same time, it has the flavor of histograms that puts similar values to the same bucket or adjacent buckets (e.g., values within a certain range are hashed to the same bucket). Consequently, LSH-sketch gains the benefit of histograms but without their limitations. Although the marriage of LSH and sketch-based

synopsis is exciting, we still need to tackle certain technicality. Specifically, LSH was originally designed for *dimension reduction*. For streaming applications, however, we might only need to maintain the sketch for a single attribute (e.g., source IP address) and the dimensionalities before and after hashing remain the same. For cases like that, a direct adoption of existing LSH implementations into the existing sketches might result in a loss of theoretical guarantees. To circumvent that, we have designed another implementation of an LSH family to regain the theoretical guarantees. Furthermore, in order to fully leverage LSH-sketch, we have also developed a corresponding range-count algorithm, namely, AOC, to adjust any over-counting when answering range-count queries using LSH-Sketch. In additional to the theoretical results about LSH-sketch and AOC, we have also carried out extensive experiments on both real data and synthetic data. Empirical results show that our method outperforms the baseline approaches in both accuracy and insertion throughput, and has query throughput on a par with the strongest baseline, under a variety of query ranges.

Chapter 2

Historic Moments Discovery in Sequence Data

In this chapter, we formally introduce the concept of *historic moment*. Section 2.1 reviews the related work. Section 2.2 gives the formal problem definition. Section 2.3 presents our space-optimal incremental algorithm together with a few baseline algorithms. Section 2.4 presents the case studies and Section 2.5 presents the experimental study. Appendix A extends the problem and the algorithms from a single data sequence to multiple data sequences.

2.1 Related Work

In this section we first review related works that focus on discovering interesting knowledge from sequence datasets. The first work is [33], which studied the finding of *prominent streaks* from a data sequence $D_n = \langle v_1, v_2, \ldots, v_n \rangle$ with n

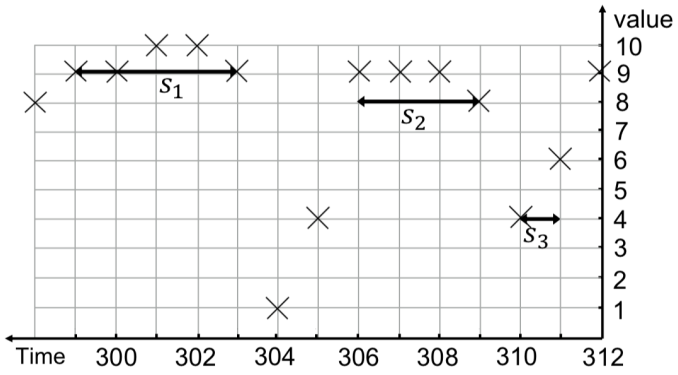


Figure 2.1: A data sequence (a value is represented by $\times$)

numeric values. A *streak* $s = (i, j, v)$ is a *contiguous subsequence* in D_n containing numeric values $\langle v_i, v_{i+1}, \ldots, v_j \rangle$, with $i \leq j$, and $v = \min \{ v_k \mid i \leq k \leq j \}$ denotes the minimum[1] value of the subsequence. We call v the *value* of s, $|s| = j - i$ the *length*[2] of s, and $[i, j]$ the *interval* of s.

Prominent streaks are in fact the 2-dimensional skyline points [2, 6, 9, 14, 41, 48, 58, 68] of all the streaks found in a sequence, based on the *length* and the *value* dimensions. That is, a streak is a prominent streak if there is no streak that has both larger length and value at the same time. The challenge of computing prominent streaks is that, given a sequence of length n, there are $\Theta(n^2)$ streaks in total. So a brute-force method would require $O(n^4)$ comparisons to locate the set of prominent streaks from $\Theta(n^2)$ candidates. In view of that, the authors in [33] developed an $O(n \log n)$ time algorithm, LLPS, which first computes a set of "*local prominent streaks*" ($\mathcal{LPS}_n$) from D_n and then locates the set of prominent streaks from $\mathcal{LPS}_n$. The size of $\mathcal{LPS}_n$ is at most n and it

[1] Maximum value is an alternate but equivalent definition.
[2] An alternate definition is $|s| = j - i + 1$.

is guaranteed that the set of prominent streaks is a subset of $\mathcal{LPS}_n$. According to [33], the definition of local prominent streak is:

Definition 1 (Local Prominent Streak (LPS) [33]). *A streak $s = (i, j, v)$ is a* local prominent streak *if (a) $v_{i-1} < v$ and (b) $v_{j+1} < v$. We use $\mathcal{LPS}_n$ to denote the set of local prominent streaks in the sequence dataset D_n. When $i = 1$, we can ignore condition (a). Similarly when $j = n$, we can ignore condition (b).*

Figure 2.1 shows some streaks of a sequence dataset. Streak $s_1(299, 303, 9)$ and $s_2(306, 309, 8)$ are local prominent streaks whereas the streak $s_3(310, 311, 4)$ **is not**. The LLPS algorithm has two phases: (1) first spend $O(n)$ time to obtain the linear size $\mathcal{LPS}_n$, (2) then invoke an $O(|\mathcal{LPS}_n| \log |\mathcal{LPS}_n|)$ skyline algorithm to compute the set of prominent streaks from $\mathcal{LPS}_n$, resulting in an overall time complexity of $O(n+|\mathcal{LPS}_n| \log |\mathcal{LPS}_n|)$, which is at most $O(n \log n)$.[3] Prominent streaks alone could only tell the continuity of a singular event. Historic moments can tell the other side of the story about how interesting that singular event is. We later show that historic moments and prominent streaks are related but computing historic moments is a challenging problem in its own right.

In [64], the authors advocated that *rare* events are more informative and *comparisons* among objects can make a story more complete. As such, given a set $\mathcal{A}$ of concerned attributes on a relational dataset, the authors developed the concept of *top-τ-skyband* as "one-of-the-few" objects. Intuitively, the top-τ skyband consists of (at most) τ objects that are dominated by the fewest number of other objects. Each of these objects is *one of the few* most prominent objects in the dataset when attributes in $\mathcal{A}$ are concerned. [64] proposed an efficient

[3] [33] also included another algorithm, NLPS, which generates $O(n^2)$ local prominent streaks in the first phase. It is obvious that LLPS is more efficient than NLPS.

algorithm that finds top-τ skyband in $O(2^d(\tau|n| + n\log n))$ time, where d is the number of attributes and n is the number of objects. Our work shares the same vision with [64] in terms of quantifying the interestingness of a piece of information through its rarity. However, we focus on sequence data whereas [64] focused on relational data.

In [57], a "fact" is defined as a *contextual skyline* object that stands out against other objects in a context with regard to a set of measures. Given a relational table with a set $\mathcal{M}$ of measure attributes and a set $\mathcal{A}$ of dimension attributes, for a constraint c defined on $A \subseteq \mathcal{A}$ (known as a context) and a measure subspace $M \subseteq \mathcal{M}$, a tuple t is a *contextual skyline object* if t satisfies c and no other tuple t' satisfying c dominates t. The authors in [57] focused on identifying these "facts" *timely*. So, when a new tuple t is added to a table, the target is to find which combinations of constraint c and measure subspace M makes t a contextual skyline object. All those eligible $\langle c, M\rangle$ pairs are treated as *situational facts* and the *prominence* of a situational fact is defined based on the size of the skyline under c and M. Upon the arrival of a new tuple t, a baseline method can compute the topmost prominent situational facts using $O(2^{|\mathcal{M}|+|\mathcal{A}|})$ skyline queries. In case the context is fixed, the number of skyline queries becomes $O(2^{|\mathcal{M}|})$. In this work, we aim to discover certain knowledge in a timely fashion as in [57]. However, other than that, our work is orthogonal with [57] because that work focused on relational data while ours focuses on sequence data.

All of the related work above rely on skyline computation, whose discussion began with [9], in which an $O(n^2)$ block nested loop (BNL) skyline algorithm was introduced. Then the sort-filter-skyline (SFS) algorithm was introduced

[14], whose best case time complexity is $O(dn + n \log n)$, where d is the number of dimensions. Index-based skyline algorithms were introduced in [37, 48, 58], in which the Branch & Bound (BBS) skyline algorithm in [49] gives the I/O optimal solution because for each query it visits relevant nodes of R-tree only once, with time complexity $O(n \log n)$. Recently, [2] presented a parallel skyline algorithm. A survey of skyline computation and the variants of skyline could be found in [34].

2.2 Problem Definition

In this thesis, we propose the notion of *historic moment*. Our goal is to identify historic moment in a **timely fashion**. So, we mainly focus on streaks that just happened, and we name those as *situational streaks*. Furthermore, inspired by our motivating example, whether a situational streak is informative or not is relative to how long ago a similar event (streak) happened before. For some applications (e.g., computation journalism), a situational streak might lead to a news story because a similar streak happened long ago. But there are also applications (e.g,. seismology) where a situational streak becomes important because a similar streak just happened. We now formally define historic moments and its related terms based on the intuition above.

Definition 2. SITUATIONAL STREAK (SS). *In a data sequence D_n with n numeric values $\langle v_1, v_2, \ldots, v_n \rangle$, a streak $(i, j, v) \in \mathcal{LPS}_n$ is a* situational streak *if $j = n$, i.e., the most recent local prominent streak. We use $\mathcal{SS}_n$ to denote the set of situational streaks in D_n.*

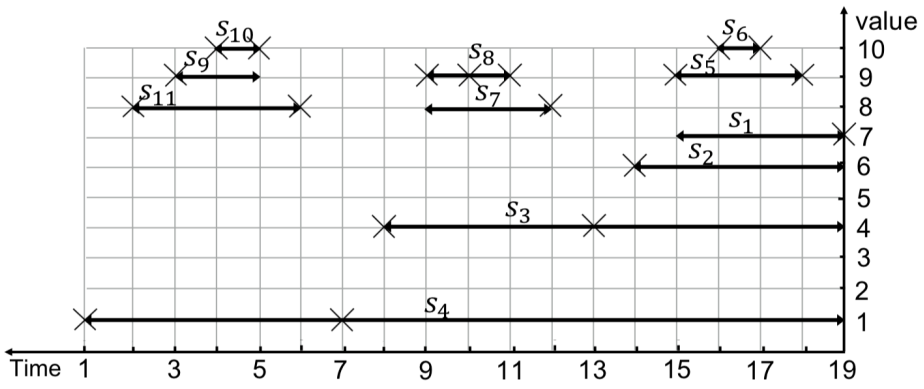


Figure 2.2: Data Sequence D_{19}

Figure 2.2 shows a data sequence D_{19} with $n = 19$ numeric values. D_{19} has four situational streaks:

$$s_1(15, 19, 7), s_2(14, 19, 6), s_3(8, 19, 4), s_4(1, 19, 1)$$

Obviously not all situational streaks are interesting and basically we are interested in those with highest or lowest values (e.g., high seismic magnitude, low Air Pollution Index). Without loss of generality, we focus on those with highest values in the discussion. Also, for ease of discussion, we assume that all values are positive. So, we define:

Definition 3. TOP-k SITUATIONAL STREAK. *The top-k situational streaks are the k streaks in $\mathcal{SS}_n$ with the highest values.*

In Figure 2.2, among the four situational streaks, s_1 and s_2 are the top-2 situational streaks.

The following news[4] is a real example of reporting not only the top but the top-2 situational streaks:

> *" ... the highest temperatures are above 32 Celsius for five days and 30 Celsius for six days ..."*

Next, given any situational streak z, we are interested in a subset of local prominent streaks that are "similar" to z. So, we define:

Definition 4. Analogous Streaks ($\mathcal{AS}$). *A local prominent streak s in a data sequence D_n is an* analogous streak *of a situational streak z when:*

1. $s.j < z.i$ *(i.e., s ends before z starts),*
2. $|s| \geq |z| \cdot \sigma$ *(i.e., the length of s is at least σ times that of z, where $\sigma \geq 0$ is a similarity threshold), and*
3. $s.v \geq z.v \cdot \sigma$ *(i.e., the value of s is at least σ times that of z).*

In Figure 2.2, s_{11}, with length 4 and value 8, is the only analogous streak of the top-1 situational streak s_1 (length 4; value 7) when the similarity threshold σ is 1. When we relax σ to be 0.75, both s_7 and s_{11} are analogous streaks of s_1.

In this thesis, we use the same similarity threshold σ for both length and value. An alternate definition is to impose different similarity thresholds on length and value, which could be easily supported by straightforward adaption of our techniques.

Definition 5. Historic Moments ($\mathcal{HM}$). *Let $\mathcal{AS}(z)$ be the set of analogous streaks of a situational streak z. Assume that each streak s in $\mathcal{AS}(z)$*

[4] http://www.chinanews.com/sh/2015/05-21/7291066.shtml

is represented by a 3D point $(|s|, s.j, s.v)$. *The* historic moments *of* z, *denoted by* $\mathcal{HM}(z)$, *is the 3D skyline of* $\mathcal{AS}(z)$, *and we write that as* $\mathcal{HM}(z) = skyline(\mathcal{AS}(z))$.

Problem Definition: *Given a data sequence* D_n *with* n *numeric values, a similarity threshold* σ, *a positive integer* k, *compute the historic moments for each of the top-k situational streaks.*

In Figure 2.2, for the top-1 situational streak s_1, if $\sigma = 0.5$, we have $\mathcal{HM}(s_1) = \{s_7, s_8, s_{11}\}$. Streak s_9 is not a historic moment of s_1 because s_8 *dominates*[5] s_9, denote as $s_8 \succ s_9$, despite $\mathcal{AS}(s_1) = \{s_7, s_8, s_9, s_{11}\}$. Note that s_7 is the historic moment of s_1 with the largest j (i.e., most recent), s_8 is the historic moment of s_1 with the highest value, and s_{11} is the historic moment of s_1 with the longest interval. In the following, we use computational journalism as an example and present some real news stories that justify the use of skyline of analogous streaks to formulate historic moments.

Story 1. [**Most recent historic moment**] The Beijing Blue Sky Days news in the introduction is a real example that illustrates that the most recent historic moment is newsworthy. In that story, the "36 days of Blue Sky" is a situational streak z with length 36 and value 100. The "30 days of Blue Sky that happened in 2008" is then the historic moment of z that *occurs most recently.*

Story 2. [**Highest value historic moment**] In April 2014, UK had the

[5]Following the literature, an object x is said to be *dominated* by another object y, denoted as $y \succ x$, with respect to a set $\mathcal{A}$ of concerned attributes, if $y.A_i \geq x.A_i$ for all $A_i \in \mathcal{A}$, and there exists at least one attribute $A_j \in \mathcal{A}$ such that $y.A_j > x.A_j$. Here we discuss upon streak s, and the concerned attributes set $\mathcal{A}$ refers to $\{|s|, s.j, s.v\}$ of s.

following news about smog:

> *"air pollution levels with more than eight lasts two days",*

which is essentially a situational streak z of length 2 with value 8. When reporting the news, the journalist quoted the "*Great Smog*" incident, which happened in 1952:

> *"...(The 1952 Smog) led to the creation of the Clean Air Act 1956, which introduced a number of measures to reduce air pollution ... Unfortunately in the modern day, despite the visibility and intensity of smog being much reduced, up to 29,000 people in the UK still die per year because of air pollution, according to the European Commission."*[6]

The "Great Smog" incident is in fact the historic moment of z *with the highest value.*

Story 3. [**Longest interval historic moment**] In January 2014, US had the following news about drought in California:

> *"'Meanwhile, today's scant rainfall was enough for Sacramento to finally end the longest running rainless rainy season streak in its recorded history. The city now has a new all-time record of 52 consecutive days without measurable rainy season rainfall,"*

[6]UK smog: You thought this was bad? Take a look at the Great Smog of 1952: `http://www.independent.co.uk/news/uk/home-news/uk-smog-you-thought-this-was-bad-take-a-look-at-the-great-smog-of-1952-9238550.html`

which is essentially a situational streak of length 52 with value close to 0. When reporting the news, the journalist continued with comparisons with a historic moment that *had the longest interval*:

> *"... dating back to Dec. 7, 2013. The previous longest streak was Nov. 1 to Dec. 16, 1884."* [7]

Table 2.1: Major Notations in Chapter 2

Notation	Meaning
D_n	Data Sequence with n values
σ	Similarity threshold
k	Top-k parameter
$s(i, j, v)$	Streak s with interval $[i, j]$ and value v
$\mathcal{LPS}_n$	Local prominent streaks of D_n
$\mathcal{SS}_n$	Situational streaks of D_n
$\mathcal{AS}(z)$	Analogous streaks of a situational streak z
$\mathcal{HM}(z)$	Historic moments of a situational streak z
$\mathcal{P}_n$	Perplexing streaks of D_n
$\mathcal{N}_n$	Non-perplexing streaks of D_n
$\mathcal{U}_n$	Minimal subset of $\mathcal{LPS}_n$ obtained by SOIA

2.3 Finding Historic Moments from a Data Sequence

In this section, we present algorithms to obtain historic moments from a data sequence. Specifically, section 2.3.1 first presents a baseline algorithm that computes historic moments from a data sequence D_n offline, given a similarity

[7] California's Devastating Drought Isn't Going to Get Better Any Time Soon
http://www.slate.com/blogs/future_tense/2014/01/30/california_s_exceptional_drought_won_t_get_better_any_time_soon.html

threshold σ and a parameter k. In practice, there could be data update or a user may want to look for historic moments with different σ and k values when operating under an interactive (online) mode [30]. In these cases, re-executing the baseline algorithm for any update of σ, k, or data would be inefficient. Therefore, in section 2.3.2, we first present how to refactor the baseline algorithm to be *incremental*. The baseline incremental algorithm is still inefficient because it needs to keep and maintain a lot of intermediate results. Therefore, in section 2.3.3, we present SOIA, an efficient incremental algorithm that returns historic moments by *maintaining and accessing minimal information*. Appendix A extends the problem and the algorithms for finding historic moments from a single data sequence to multiple data sequences. Table 2.1 lists the major notations used in this chapter.

2.3.1 Baseline Algorithm (BA)

Given a data sequence D_n, a similarity threshold σ, and a parameter k, the problem of finding historic moments for each of the top-k situational streaks can be solved naively as follows:

- Step 1. Use the first phase of LLPS in [33] to compute the set of all local prominent streaks $\mathcal{LPS}_n$ from D_n, and then select the top-k situational streaks from there. This step, according to [33], takes $O(n)$ time and the $\mathcal{LPS}_n$ takes $O(n)$ space.

- Step 2. For each top-k situational streak z, scan through $\mathcal{LPS}_n$ to identify its analogous streaks $\mathcal{AS}(z)$.

- Step 3. Finally, for each z, compute the skyline from $\mathcal{AS}(z)$ as the resulting $\mathcal{HM}(z)$.

The naive method above is inefficient in terms of full scanning the large $\mathcal{LPS}_n$ k times in Step 2. Therefore, one simple improvement is build an index for $\mathcal{LPS}_n$ after Step 1. Specifically, one can regard each streak s in $\mathcal{LPS}_n$ as a 3D point $(|s|, s.j, s.v)$, and insert them into an R-tree. Then, the analogous streaks of a situational streak z can be regarded as a 3D-range query Q:

$$[|z| \cdot \sigma, +\infty) \times [0, z.i) \times [z.v \cdot \sigma, +\infty)$$

By building an R-tree, the above query can locate the analogous streaks of a situational streak z without scanning all the streaks in $\mathcal{LPS}_n$. Furthermore, Steps 2 and 3 can be combined as a constrained skyline query on the R-tree that returns the skyline within Q. This combined step can be implemented using the BBS skyline algorithm in [49]. Algorithm 1 summarizes the above index based baseline algorithm, namely, algorithm BA.

Algorithm 1 Baseline Algorithm (BA)

```
1: procedure BA(D_n, σ, k)
2:     Use the first phase of LLPS in [33] to compute LPS_n;
3:     Rtree = Build-R-Tree(LPS_n);
4:     SS_n = Get-SS(LPS_n);
5:     Z = Get-Top-SS(SS_n, k);
6:     for each streak z in Z do
7:         Q = [|z| · σ, +∞) × [0, z.i) × [z.v · σ, +∞); // define the analogous region
8:         HM(z) = BBS(Rtree, Q); // compute constrained skyline
```

2.3.2 Baseline Incremental Algorithm (BIA)

The baseline algorithm (BA) is not incremental. Therefore, we refactor it to become incremental so that it can return results efficiently even when the data sequence D_n is appended with new values resulting in D_m, where $m > n$, or when the parameters are updated as σ' or k'. The incremental method is composed of two phases where there are (`i`) a MAINTENANCE procedure to compute $\mathcal{LPS}_m$ online when data is appended, and (`ii`) a LOOKUP procedure to answer the historic moment queries. Similar to BA, we use an R-tree to store the local prominent streaks.

2.3.2.1 BIA Maintenance

When the data sequence D_n is appended with new values $\langle v_{n+1}, v_{n+2}, \cdots, v_m \rangle$ resulting in D_m, where $m > n$, this procedure aims to obtain (i) the updated set of situational streaks $\mathcal{SS}_m$ and (ii) possibly some new local prominent streaks. The MAINTENANCE procedure is similar to the one in [33]. Specifically, for each value $v_{n+k} \in \{v_{n+1}, v_{n+2}, \cdots, v_m\}$, where $1 \leq k \leq m - n$, it may:

1. make some streaks in $\mathcal{SS}_{n+k-1}$ to stop being situational streaks and turn to being local prominent streaks that ends at $n + k - 1$. Those streaks would then get inserted into the R-tree;
2. extend some streaks in $\mathcal{SS}_{n+k-1}$ to become longer streaks in $\mathcal{SS}_{n+k}$ that end at $n + k$.
3. form a new local prominent streak, whose value is v_{n+k} and ends at $n + k$;

this happens when none of the streak in $\mathcal{SS}_{n+k-1}$ has value v_{n+k}. Note that this is a situational streak for D_{n+k}, so it is in $\mathcal{SS}_{n+k}$.

Algorithm 2 BIA Maintenance Procedure

```
1: procedure MAINTENANCE(⟨v_{n+1}, v_{n+2} ··· v_m⟩)
2:     for k = 1 to m − n do
3:         if n == 0 and k == 1 then
4:             SS_1 = {(1, 1, v_1)};
5:             continue;
6:         SS_{n+k} = ∅;
7:         for each streak (i, n + k − 1, v) in SS_{n+k−1} do
8:             if v_{n+k} ≥ v then
9:                 Insert (i, n + k, v) to SS_{n+k};                    //case (2)
10:            else
11:                Insert (i, n + k − 1, v) to Buffer B;                // case (1)
12:        if no streak in SS_{n+k−1} has value v_{n+k} then            // case (3)
13:            if all streaks in SS_{n+k−1} have value < v_{n+k} then
14:                Insert (n + k, n + k, v_{n+k}) to SS_{n+k};
15:            else
16:                Select the streak (i, n + k − 1, v) in SS_n whose value v > v_{n+k}
   and is the smallest;
17:                Extend it to be (i, n + k, v_{n+k})
18:                Insert it into SS_{n+k};
19:    Insert the streaks in B into Rtree;
```

For maintenance reason that becomes clear momentarily, we separate $\mathcal{SS}_n$ with the rest of streaks in $\mathcal{LPS}_n$. Specifically, we insert streaks from $\mathcal{LPS}_n - \mathcal{SS}_n$ as 3D points into an R-tree. Streaks from $\mathcal{SS}_n \subseteq \mathcal{LPS}_n$ are stored separately. Algorithm 2 summarizes the above discussion, and we have:

- The MAINTENANCE procedure takes $\mathcal{SS}_n$ and the appending values $\langle v_{n+1}, v_{n+2}, \cdots, v_m \rangle$ as the input. The outputs of it are $\mathcal{SS}_m$ and the updated R-tree by inserting new local prominent streaks.
- For each value $v_{n+k} \in \{v_{n+1}, v_{n+2}, \cdots, v_m\}$, Lines 2–18 computes its $\mathcal{SS}_{n+k}$

and the new local prominent streaks based on $\mathcal{SS}_{n+k-1}$ iteratively. Lines 3–5 are to initialize $\mathcal{SS}_1$.

- For each value $v_{n+k} \in \{v_{n+1}, v_{n+2}, \cdots, v_m\}$, we collect the new local prominent streaks in a Buffer B (Line 11) and then insert them into the R-tree in a batch (Line 19) to reduce the I/O overhead by avoiding frequent access to the R-tree.

2.3.2.2 BIA Lookup

Given the R-tree of historic moments candidates created by the MAINTENANCE step, the historic moments for each of the top-k situational streaks, under any similarity parameters σ' and k' value, can be obtained by calling the LOOKUP procedure as presented in Algorithm 3.

Algorithm 3 BIA Lookup Procedure

```
1: procedure LOOKUP(σ', k')
2:     Z = Get-Top-SS(SS_n, k');
3:     for each streak z in Z do
4:         Q = [|z| · σ', +∞] × [0, z.i) × [z.v · σ', +∞];
5:         HM(z) = BBS(Rtree, Q); // compute constrained skyline
```

2.3.2.3 Space requirement of BIA

Why is it necessary for BIA to keep all streaks in $\mathcal{LPS}_n$? Specifically, from what we have defined, the historic moments are the *skyline* of analogous streaks, which are in turn a *subset* of $\mathcal{LPS}_n$. Therefore, one might question whether BIA can keep only the skyline of $\mathcal{LPS}_n$, instead of the full $\mathcal{LPS}_n$. Unfortunately, optimizing BIA like that is incorrect. That is because a streak currently not in

the $skyline(\mathcal{LPS}_n)$ could become a historic moment after a new value v_{n+1} is appended, or when facing different values of σ and k. The followings are two examples.

Example 1. *Consider again the data sequence in Figure 2.2, we have:*

- $n = 19$;
- $\mathcal{LPS}_{19} = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}\}$;
- $\mathcal{SS}_{19} \subset \mathcal{LPS}_{19} = \{s_1, s_2, s_3, s_4\}$;

Consider the skyline of $\mathcal{LPS}_{19}$, which is:

- $skyline(\mathcal{LPS}_{19}) = \{s_1, s_2, s_3, s_4, s_5, s_6, s_{11}\}$

Now consider the lookup of historic moment of Top-1 situational streak in that dataset D_{19} with $\sigma' = 0.75$. The top-1 situational streak in that dataset D_{19} is s_1 whereas the analogous streaks of s_1 are $\mathcal{AS}(s_1) = \{s_7, s_{11}\}$. The historic moments of s_1 are $\mathcal{HM}(s_1) = \{s_7, s_{11}\}$, since s_7 and s_{11} cannot dominate each other. At this point, we see that s_7 is a historic moment of s_1 but $s_7 \notin skyline(\mathcal{LPS}_{19})$.

Example 2. *Still consider the data sequence in Figure 2.2, but with a new value $v_{20} = 5$ appended, resulting in D_{20} of Figure 2.3. We see that, with v_{20} appended, it has*

1. *made streaks s_1 and s_2 in $\mathcal{SS}_{19}$ to stop being situational streaks and turn to being local prominent streaks that ends at $n = 19$.*

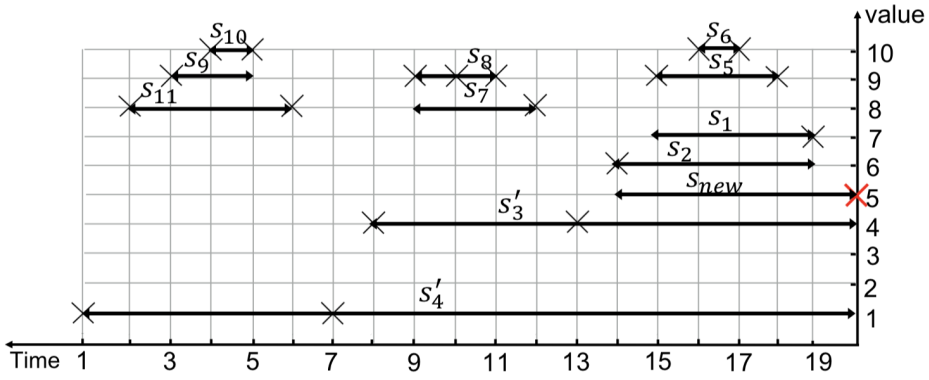


Figure 2.3: After v_{20} is appended

2. *extended streaks* s_3 *and* s_4 *in* $\mathcal{SS}_{19}$ *to become longer streaks* s'_3 *and* s'_4 *in* $\mathcal{SS}_{20}$ *that ends at* $n = 20$.

3. *formed a new local prominent streak,* s_{new}*, whose value is* 5 *and ends at* $n = 20$.

Now consider the lookup of historic moment of Top-1 situational streak in that updated dataset D_{20} *with* $\sigma' = 0.5$*. The top-1 situational streak in that dataset* D_{20} *is* s_{new} *whereas the analogous streaks of* s_{new} *are* $\mathcal{AS}(s_{new}) = \{s_7, s_{11}\}$*. The historic moments of* s_{new} *are* $\mathcal{HM}(s_{new}) = \{s_7, s_{11}\}$*. Once again, we see that* s_7 *is a historic moment of* s_{new} *but* $s_7 \notin skyline(\mathcal{LPS}_{19})$.

2.3.3 Space Optimal Incremental Algorithm (SOIA)

BIA needs to keep and maintain $\mathcal{LPS}_n$, which is space inefficient, especially when multiple sequences are of interest (e.g., multiple stocks, multiple seismic monitoring sensors), or when the sequences are very long (e.g., high-frequency trading with stock tick every millisecond). As showed in the previous sections,

straightforward optimizations like keeping $skyline(\mathcal{LPS}_n)$, instead of $\mathcal{LPS}_n$, is unfortunately incorrect. When not space efficient, BIA would not be time efficient either because of the redundancies in the index would jeopardize both the lookup and maintenance time. In this section, we present a space optimal incremental algorithm (SOIA) that keeps only a minimal subset $\mathcal{U}_n$ of $\mathcal{LPS}_n$ that is sufficient to return historic moments online. Formally,

Definition 6. *Given a data sequence D_n, the* MINIMAL SUBSET *$\mathcal{U}_n$ of $\mathcal{LPS}_n$ refers to a subset of $\mathcal{LPS}_n$ that guarantees*

1. *for any streak s in $\{\mathcal{LPS}_n - \mathcal{U}_n\}$, it must not be a historic moment of any situational streak y in D_m, where $m \geq n$.*

2. *there does* **NOT** *exist a proper subset X of $\mathcal{U}_n$ that satisfies condition (1).*

SOIA strikes to maintain $\mathcal{U}_n$ under continuous data updates. Space-optimality of SOIA significantly reduces the size of the index, thereby reducing the I/O per operation or making the index memory-resident. That property is crucial for online analysis and real-time monitoring, especially when there are possibly many data sequences of interest, which demands one index per data sequence.

So the grand challenge of SOIA is how to confine $\mathcal{U}_n$, where trivially confining $\mathcal{U}_n$ to be the skyline of $\mathcal{LPS}_n$ is definitely insufficient, whereas confining $\mathcal{U}_n$ to be $\mathcal{LPS}_n$ is definitely not space optimal. Now we first discuss how to obtain $\mathcal{U}_n$ properly given the data sequence D_n, and section 2.3.3.1 studies how to maintain its minimality when the data sequence is appended.

In SOIA, we treat σ, as a parameter that controls the tradeoff between space and time, in addition to its original role as being the similarity threshold. More specifically, when deciding which streaks in $\mathcal{LPS}_n$ shall be kept (i.e., belong to $\mathcal{U}_n$) and which shall be discarded, a small σ value makes local prominent streaks easier to be an analogous streak of a situational streak z. So, it is natural that the size of $\mathcal{U}_n$ increases when σ is getting smaller. In contrast, a large σ value makes local prominent streaks harder to be an analogous streak of a situational streak z. So, it is intuitive that the size of $\mathcal{U}_n$ decreases when σ is getting larger. With σ as a parameter between space and time, the goal of SOIA is to maintain $\mathcal{U}_n$ as long as a user queries for historic moments with any similarity value σ' larger than or equal to σ.

We now confine what $\mathcal{U}_n$ should be, given σ. We start by showing a simple lemma (Lemma 1). Then, we look at the easiest case, with $\sigma = 1$. We can interpret this case as either having the most stringent space requirement, or as the users being uninterested in analogous streaks that are shorter, or smaller in value, than the situational streak of interest. Finally, we work on the general case.

Lemma 1. *The intervals of two local prominent streaks are either disjoint, or one containing the other.*

Proof. Assume to the contrary that there exist two local prominent streaks (i, j, v) and (i', j', v') whose intervals are not disjoint, nor one containing the other. Without loss of generality, assume that $i < i'$. Then, we have $i < i' \leq j < j'$. That is, $i' - 1$ is within the range of $[i, j]$ and $j + 1$ is within the range of $[i', j']$. Now, there are two cases:

1. If $v \geq v'$, then (i', j', v') is not an local prominent streak because $v_{i'-1} \geq v \geq v'$.
2. Otherwise, we have $v' > v$. Then (i, j, v) is not an local prominent streak because $v_{j+1} \geq v' > v$.

So, both cases lead to contradiction. The lemma follows. □

The Specific Case: What streaks in $\mathcal{LPS}_n$ should be in $\mathcal{U}_n$ when $\sigma = 1$?

Let $\mathcal{HM}(\mathcal{SS}_n)$ denote the set that contains the historic moments of all streaks in $\mathcal{SS}_n$ and when $\sigma = 1$ we have the following proposition:

Proposition 1. *When $\sigma = 1$, $skyline(\mathcal{LPS}_n) \cup \mathcal{HM}(\mathcal{SS}_n)$ is the minimal subset $\mathcal{U}_n$ of $\mathcal{LPS}_n$ that contains historic moments of situational streak for dataset D_n or future dataset D_m, where $m > n$.*

Proof. We prove by showing that a streak s is in $skyline(\mathcal{LPS}_n)$ or $\mathcal{HM}(\mathcal{SS}_n)$ if and only if s can serve as a historic moment of some streaks in $\mathcal{SS}_n$ or $\mathcal{SS}_m$. This is done by proving Lemma 2 (the if case) and Lemma 3 (the only if case) below.

Lemma 2. *If a streak s in $\mathcal{LPS}_n$ can serve as a historic moment of a streak in $\mathcal{SS}_n$ or $\mathcal{SS}_m$, then s is in $skyline(\mathcal{LPS}_n) \cup \mathcal{HM}(\mathcal{SS}_n)$.*

. It is equivalent to showing that for any streak s, if $s \notin$

$skyline(\mathcal{LPS}_n)$ and $s \notin \mathcal{HM}(\mathcal{SS}_n)$, then s will not be a historic moment of any streak in $\mathcal{SS}_n$ or $\mathcal{SS}_m$.

Since s is not in the skyline of $\mathcal{LPS}_n$, there must exist some streak $y \in \mathcal{LPS}_n$ in the skyline with $y \succ s$. Now, for any streak $z^* = (i^*, j^*, v^*)$ in $\mathcal{SS}_n$ or $\mathcal{SS}_m$,

1. If $z^* \in \mathcal{SS}_n$, s cannot be a historic moment of z^* since $s \notin \mathcal{HM}(\mathcal{SS}_n)$.
2. Else if y does not overlap with z^*, y will be an analogous streak of z^* whenever s is, so that s cannot be in the skyline of $\mathcal{AS}(z^*)$, and thus not a historic moment of z^*.
3. Else, y overlaps with z^* and $z^* \notin \mathcal{SS}_n$. Then, we must have some $z = (i, j, v)$ in $\mathcal{SS}_n$ with the same starting position as z^*, i.e., $i = i^*$, and also y must overlap with z. Also, $|z| < |z^*|$ since $z^* \notin \mathcal{SS}_n$ while both streaks z and z^* have the same starting position. Now, recall that from Lemma 1, the intervals of two local prominent streaks are either disjoint, or one containing the other. Thus, we further have $|s| \leq |y|$ (since $y \succ s$) and $|y| \leq |z|$ (since y overlaps with z and $z \in \mathcal{SS}_n$). The above inequalities imply that $|s| < |z^*|$, so that s is not an analogous streak, and thus not a historic moment of z^* when $\sigma = 1$.

So in all cases, s is not a historic moment of z^*. This completes the proof of the lemma.

■

Lemma 3. *If a streak s is in $skyline(\mathcal{LPS}_n) \cup \mathcal{HM}(\mathcal{SS}_n)$, then s can serve as a historic moment of $\mathcal{SS}_n$ or $\mathcal{SS}_m$.*

. Since any streak in $\mathcal{HM}(\mathcal{SS}_n)$ is already a historic moment of a current situational streak, it remains to show that any streak $s = (i, j, v)$ in $skyline(\mathcal{LPS}_n)$ can be a historic moment of some streak in $\mathcal{SS}_n$ or $\mathcal{SS}_m$. To see this, imagine that the data sequence D_n is appended with the following values for its next $|s| + 2$ days: $\epsilon, v, v, \ldots, v$, where ϵ is a positive value less than v. That essentially will create a situational streak $z^* = (i^*, j^*, v^*)$, with $i^* = n + 2$, $j^* = n + |s| + 2$, length $|z^*| = j^* - i^* = |s|$ and value $v^* = v$.

Note that s is an analogous streak of z^*. Furthermore, s is in the skyline of $\mathcal{AS}(z^*)$ because (i) every streak in $\mathcal{AS}(z^*)$ must end before $n + 1$, but (ii) every streak that ends exactly at $n + 1$ will have minimum value ϵ, which cannot be an analogous streak of z^*. This implies that $\mathcal{AS}(z^*) \subseteq \mathcal{LPS}_n$. So, no streak in $\mathcal{AS}(z^*)$ dominates s as $s \in skyline(\mathcal{LPS}_n)$. Thus, s is in the skyline of $\mathcal{AS}(z^*)$, and is therefore a historic moment of z^*. This completes the proof of the lemma. ■

By combining Lemmas 2 and 3, Proposition 1 follows. □

For Figure 2.2 as in Example 1, we have

- $\mathcal{SS}_{19} = \{s_1, s_2, s_3, s_4\}$
- $skyline(\mathcal{LPS}_{19}) = \{s_1, s_2, s_3, s_4, s_5, s_6, s_{11}\}$.

When we set $\sigma = 1$ and $k = 4$, the historic moments of $\mathcal{SS}_{19}$, denoted as $\mathcal{HM}(\mathcal{SS}_{19})$, is $\{s_{11}\}$, since s_{11} is a historic moment (and the only one) of s_1, while there are no historic moments for s_2, s_3, or s_4. So, Proposition 1 states that we only need to keep $\{s_1, s_2, s_3, s_4, s_5, s_6, s_{11}\}$. Although currently s_5 or s_6 are not historic moments of any situational streak (since they overlap with all streaks in $\mathcal{SS}_{19}$), each of them may be a historic moment of some situational streaks in the future. For example, consider two new values $v_{20} = v_{21} = 9$ are added to the example data sequence in Figure 2.2. Then, $(20, 21, 9)$ becomes a situational streak of the data sequence D_{21}, with both s_5 and s_6 being its historic moments.

Example 3. *Consider v_2 and v_6 equal to 7 instead of 8 in Figure 2.2. In this case, streak $s_{11} = (2, 6, 7)$ is no longer in skyline($\mathcal{LPS}_{19}$), since it is dominated by s_1. Yet, s_{11} is a historic moment of s_1, and $s_{11} \in \mathcal{HM}(\mathcal{SS}_{19})$.*

The example above illustrates why Proposition 1 has to keep $\mathcal{HM}(\mathcal{SS}_n)$ as well. In that example, $s_{11} \in \mathcal{HM}(\mathcal{SS}_{19})$. This also justified our aforementioned argument of why modifying BIA to keep only the skyline of $\mathcal{LPS}_n$ is insufficient.

The General Case: What streaks in $\mathcal{LPS}_n$ should be in $\mathcal{U}_n$ when $\sigma > 0$?

Unfortunately, Proposition 1 is still insufficient, specifically when there are queries with $\sigma' < 1$. As we discussed in Example 1, in Figure 2.2, when querying with $\sigma' = 0.75$, s_7 is the historic moment of s_1, but $s_7 \notin \mathit{skyline}(\mathcal{LPS}_{19}) \cup \mathcal{HM}(\mathcal{SS}_{19})$ according to Proposition 1.

So, to support a general σ, in addition to $skyline(\mathcal{LPS}_n) \cup \mathcal{HM}(\mathcal{SS}_n)$, what else shall we include while maintaining minimality?

We answer the question by continuing Example 1. Specifically, when $\sigma = 0.75$, s_7 shall be included but unfortunately it is *pruned* by s_5 because $s_5 \succ s_7$. However, while s_7 is pruned by s_5, the latter is not serving as an analogous streak of s_1 instead. Why s_5 cannot serve as s_1's analogous streak? That is because s_5 *overlaps* with s_1, which violates the definition of analogous streak (Definition 4 condition 1, i.e., s_5 has to end before s_1 starts). In other words, when s_5 cannot serve as an analogous streak of s_1, it shall not be used to prune any analogous streak for s_1.

So, we classify the streaks in $\mathcal{LPS}_n$ into two subsets: (i) *Perplexing Streaks* $\mathcal{P}_n$, and (ii) *Non-perplexing Streaks* $\mathcal{N}_n$:

Definition 7. PERPLEXING STREAKS $\mathcal{P}_n$. *A streak $p \in \mathcal{LPS}_n$ is a* perplexing *streak when there exists a situational streak $z \in \mathcal{SS}_n$ such that:*

1. *p overlaps with z,*
2. *$|p| \geq |z| \cdot \sigma$ and*
3. *$p.v \geq z.v \cdot \sigma$.*

Note that p can overlap with multiple situational streaks. We use $\mathcal{P}_n$ to denote the set of all perplexing streaks in data sequence D_n.

By definition, $\mathcal{SS}_n \subseteq \mathcal{P}_n$.

Definition 8. NON-PERPLEXING STREAKS $\mathcal{N}_n$. *Streaks in $\mathcal{LPS}_n$ that are not in $\mathcal{P}_n$ are* non-perplexing *streaks, denoted as $\mathcal{N}_n$. That is, $\mathcal{N}_n = \mathcal{LPS}_n - \mathcal{P}_n$.*

So, a streak $s \in \mathcal{LPS}_n$ is a non-perplexing streak if, for every situational streak $z \in \mathcal{SS}_n$ that s overlaps with, either $|s| < |z| \cdot \sigma$ or $s.v < z.v \cdot \sigma$.

Example 4. *In Figure 2.2, when $\sigma = 0.75$, streaks in $\mathcal{LPS}_{19} = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}\}$ are classified into*

- $\mathcal{P}_{19} = \{s_1, s_2, s_3, s_4, s_5\}$
- $\mathcal{N}_{19} = \{s_6, s_7, s_8, s_9, s_{10}, s_{11}\}$.

s_5 is in $\mathcal{P}_{19}$ because (a) s_5 overlaps with s_1, (b) $|s_5| \geq |s_1| \cdot 0.75$, and (c) $s_5.v \geq s_1.v \cdot 0.75$. In contrast, $s_6 \notin \mathcal{P}_{19}$ because $|s_6| \not\geq |s_i| \cdot 0.75$ for any $s_i \in \mathcal{SS}_{19}$.

In the following, we state some important lemmas.

Lemma 4. *A streak that is dominated by skyline($\mathcal{N}_n$) would not be a historic moment of any streak in either $\mathcal{SS}_n$ for dataset D_n, or $\mathcal{SS}_m$ for future dataset D_m, where $m > n$.*

Proof. Suppose that streak s is dominated by a streak $y \in skyline(\mathcal{N}_n)$. Then, for any streak z^* in $\mathcal{SS}_n$ or $\mathcal{SS}_m$, two cases can happen: (i) y does not overlap with z^*, or (ii) y overlaps with z^*.

In Case (i), if s is an analogous streak of z^*, so is y, as $y \succ s$, so that s is not in the skyline of $\mathcal{AS}(z^*)$, and thus not a historic moment of z^*.

In Case (ii), let $z^* = (i^*, j^*, v^*)$. Since y overlaps with z^*, we must have some $z = (i, j, v)$ in $\mathcal{SS}_n$ with the same starting position as z^*, i.e., $i = i^*$, and also y overlaps with z. (Note that $z = z^*$ if $z^* \in \mathcal{SS}_n$.) As $y \in \mathcal{N}$, $|y| < |z|\,\sigma \leq |z^*|\,\sigma$ holds. As $y \succ s$, this further implies $|s| \leq |y| < |z^*|\,\sigma$. Thus, s is not an

analogous streak of z^*. Consequently, z cannot be a historic moment of z^*. This completes the proof. □

Lemma 4 directly leads to the following corollary.

Corollary 1. *Given a streak $s \in \mathcal{N}_n$, if $s \in \mathcal{HM}(z)$, where $z \in \mathcal{SS}_n \cup \mathcal{SS}_m$, then $s \in skyline(\mathcal{N}_n)$.*

Definition 9. SMALLEST-VALUE EXTENSION.

Let $z = (i, j, v)$ be a situational streak. Suppose that the sequence is appended with a new value $v' < v$. If the streak $z' = (i, j + 1, v')$ remains as a streak in $\mathcal{LPS}_{n+1}$ (i.e., $v_{i-1} < v'$), then z' is called the v' extension of z. If v_{small} is the smallest value v' such that the v' extension of z exists, then the streak $z^+ = (i, j + 1, v_{small})$ is called the smallest-value extension of z.

So for Figure 2.2, the corresponding smallest-value extensions of the situational streaks are:

$$s_1^+ = (15, 20, 6 + \epsilon) \quad s_2^+ = (14, 20, 4 + \epsilon)$$
$$s_3^+ = (8, 20, 1 + \epsilon) \quad s_4^+ = (1, 20, 0 + \epsilon)$$

where ϵ is an arbitrarily small positive value. Note that streak $(15, 20, 7)$ is not s_1^+ because 7 is not the smallest possible value to extend $s_1 = (15, 19, 7)$. Appending the data sequence with $v_{small} = 6$ will result in a situational streak $(14, 20, 6)$. But that does not qualify as s_1^+ because the starting position of s_1^+ should be the same as the starting position of s_1.

Definition 10. UNIVERSAL DOMINATION. *Let $\mathcal{SS}_n^{(p)}$ be the set of situational*

streaks that overlap with a perplexing streak p. A streak s is universally dominated *by a perplexing streak p when*

1. $p \succ s$, *and*
2. s *is not an analogous streak of all* $z \in \mathcal{SS}_n^{(p)}$, *and*
3. s *is not an analogous streak of* z^+, *for all* $z \in \mathcal{SS}_n^{(p)}$.

Conceptually, universal domination is harder to achieve than the ordinary domination, since p universally dominates s implies p dominates s.

Lemma 5. *A streak* $s \in skyline(\mathcal{N}_n) \cup \mathcal{P}_n$ *that is either (i)* universally dominated *by some streak in* $\mathcal{P}_n$ *or (ii) its length is less than* σ, *then* s *would not be a historic moment of any streak in* $\mathcal{SS}_n$ *for dataset* D_n *or* $\mathcal{SS}_m$ *for future dataset* D_m, *where* $m > n$.

Proof. 1. Suppose that a streak s is universally dominated by a perplexing streak $p = (i', j', v') \in \mathcal{P}_n$. Then, there are the following cases when considering a current/future situational streak $z^* = (i^*, j^*, v^*)$:

   (a) z^* does not overlap with p. Then, if s is analogous streak of z, so is p. In this case, s is not in the skyline of $\mathcal{AS}(z^*)$, and thus not a historic moment of z^*.

   (b) z^* overlaps with p.

      i. $z^* \in \mathcal{SS}_n$: then $z^* \in \mathcal{SS}_n^{(p)}$, so that by definition s cannot be an analogous streak of z^*.

      ii. $z^* \notin \mathcal{SS}_n$ (it happens when $z^* \in \mathcal{SS}_m$ and $z^* \notin \mathcal{SS}_n$): then there exists some current situational streak $z = (i, j, v)$ with the same

starting position as z^*, and $z \in \mathcal{SS}_n^{(p)}$. Furthermore, $|z^*| \geq |z^+|$ and $z^*.v \geq z^+.v$ hold, where z^+ is the smallest-value extension of z. As s is not an analogous streak of z^+, s cannot be an analogous streak of z^*. In other words, s cannot be a historic moment of z^*.

2. Any streak in $\mathcal{SS}_n$ or $\mathcal{SS}_m$ has length at least 1. So for streaks in $skyline(\mathcal{N}_n) \cup \mathcal{P}_n$ with length less than σ cannot be the historic moment, as its length is not qualified.

This completes the proof. □

Lemma 5 leads to the following corollary.

Corollary 2. *Given a streak $s \in skyline(\mathcal{N}_n) \cup \mathcal{P}_n$, if $s \in \mathcal{HM}(z)$, where $z \in \mathcal{SS}_n \cup \mathcal{SS}_m$, then $\forall p \in \mathcal{P}_n$ such that s is not universally dominated by p, and s has length no less than σ.*

Example 5. *Consider Figure 2.2 again, recall that when $\sigma = 0.75$, $\mathcal{P}_{19} = \{s_1, s_2, s_3, s_4, s_5\}$, $\mathcal{N}_{19} = \{s_6, s_7, s_8, s_9, s_{10}, s_{11}\}$. Also, $skyline(\mathcal{N}_{19}) = \{s_6, s_7, s_8, s_{11}\}$. Then, s_5 is a perplexing streak. Note that $s_8 \in skyline(\mathcal{N}_{19})$ and s_8 is universally dominated by s_5 since:*

1. *$s_5 \succ s_8$,*
2. *s_8 is not an analogous streak of any streak in $\mathcal{SS}_{19}^{(s_5)} = \{s_1, s_2, s_3, s_4\}$ and*
3. *s_8 is not an analogous streak of s_1^+, s_2^+, s_3^+, s_4^+.*

Therefore, by Lemma 5, s_8 would not be a historic moment in any case and it is not necessary to keep it. In contrast, $s_7 \in skyline(\mathcal{N}_{19})$ and $s_5 \succ s_7$ as well.

However, since s_7 is an analogous streak of s_1, s_7 is not universally dominated by s_5 and it cannot be pruned using Lemma 5.

Theorem 1. *The minimal subset $\mathcal{U}_n$ of $\mathcal{LPS}_n$ is the set of streaks in $skyline(\mathcal{N}_n) \cup \mathcal{P}_n$ that (i) are not universally dominated by any streak in $\mathcal{P}_n$ and (ii) have length at least σ.*

Proof. We prove the theorem by showing that a streak s is in $\mathcal{U}_n$ if and only if s can serve as a historic moment of some streak in $\mathcal{SS}_n$ or $\mathcal{SS}_m$, where $m > n$. This is done by proving Lemma 6 (the if case) and Lemma 7 (the only if case) below.

Lemma 6. *If a streak s can serve as a historic moment of some streak in $\mathcal{SS}_n$ or $\mathcal{SS}_m$, then s is in $\mathcal{U}_n$.*

Proof. Firstly, $\mathcal{LPS}_n = \mathcal{N}_n \cup \mathcal{P}_n$ contains all local prominent streaks, and thus all historic moments. Then, by Corollary 1, we see that $skyline(\mathcal{N}_n) \cup \mathcal{P}_n$ contains all historic moments. Consequently, by Corollary 2, we see that $\mathcal{U}_n$ contains all historic moments. The lemma follows. □

Lemma 7. *If a streak s is in $\mathcal{U}_n$, then s can serve as a historic moment of some streak in $\mathcal{SS}_n$ or $\mathcal{SS}_m$.*

Proof. A streak s in $\mathcal{U}_n$ is either (i) not dominated by any other streak in $\mathcal{LPS}_n$, or (ii) dominated only by some streak $p \in \mathcal{P}_n$ but not universally dominated by p.

For Case (i), if we append the data sequence D_n first with an arbitrarily small positive value ϵ, followed by $\lfloor |s|/\sigma \rfloor + 1$ values of $(s.v)/\sigma$, then, after $\lfloor |s|/\sigma \rfloor + 2$ new data values arrived, there will be a situational streak z^* with length $|z^*| = \lfloor |s|/\sigma \rfloor$ and value $z.v = (s.v)/\sigma$. That streak z^* will regard s as its historic moment.[8]

For Case (ii), let z be the longest situational streak in $\mathcal{SS}_n$ such that s is an analogous streak of z^*, where $z^* = z$ or z^+. Note that such a z must exist since s is not universally dominated by some $p \in \mathcal{P}_n$ (Recall Definition 10 for the property of universal domination). Also, $z^* \in \mathcal{SS}_n \cup \mathcal{SS}_m$.

Then, for any streak $p \in \mathcal{P}_n$ with $p \succ s$, p must overlap with z^*.[9] So, all streaks in $\mathcal{P}_n$ that dominate s cannot be an analogous streak of z^*. Moreover, no streak in $\mathcal{N}_n$ dominates s, as s is dominated only by streaks in $\mathcal{P}_n$. The above statements imply that s is not dominated by any analogous streak of z^*, so that it is in the skyline of $\mathcal{AS}(z^*)$, and thus a historic moment of z^*. In summary, each streak in $\mathcal{U}_n$ is a historic moment of some streak in $\mathcal{SS}_n$ or $\mathcal{SS}_m$, and therefore needs to be kept. □

Combining the above lemmas, Theorem 1 follows. □

[8] The reason is as follows. Clearly, s is an analogous streak of z^*. Next, if to the contrary that s is not in the skyline of $\mathcal{AS}(z^*)$, then there is some streak $y \in \mathcal{AS}(z^*)$ that dominates s. Such a y must not overlap with z^* (since it is an analogous streak of z^*), and must not include the value ϵ that is newly appended to D (since the value of y is at least $s.v$ for $y \succ s$). In other words, y must be a streak in $\mathcal{LPS}_n$. A contradiction occurs, for no streak in $\mathcal{LPS}_n$ should dominate s.

[9] Else, $\mathcal{SS}_n^{(p)}$ does not contain any situational streak of which s is an analogous streak, so that s is universally dominated by p, and therefore $s \notin \mathcal{U}_n$. A contradiction occurs.

Example 6. *Consider Figure 2.2 when $\sigma = 0.75$, as in Example 5, we have*

- $\mathcal{P}_{19} = \{s_1, s_2, s_3, s_4, s_5\}$,
- $skyline(\mathcal{N}_{19}) = \{s_6, s_7, s_8, s_{11}\}$,
- *In $skyline(\mathcal{N}_{19}) \cup \mathcal{P}_{19}$, s_8 is universally dominated by s_5, and*
- *each streak in $skyline(\mathcal{N}_{19}) \cup \mathcal{P}_{19}$ has length at least σ.*

According to Theorem 1, minimal subset $\mathcal{U}_{19}$ of $\mathcal{LPS}_{19}$ is the set of streaks in $skyline(\mathcal{N}_{19}) \cup \mathcal{P}_{19}$ that (i) are not universally dominated by any streak in $\mathcal{P}_{19}$, which gives $s_8 \notin \mathcal{U}_{19}$; (ii) have length at least σ. So we have

- $\mathcal{U}_{19} = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_{11}\}$, *and*
- $\mathcal{SS}_{19} = \{s_1, s_2, s_3, s_4\}$

2.3.3.1 SOIA Maintenance

Theorem 1 gives how to obtain $\mathcal{U}_n$ of data sequence D_n. Based on that, now we discuss how to obtain $\mathcal{U}_m$ accordingly when new values $\langle v_{n+1}, v_{n+2}, \ldots, v_m \rangle$ are appended to the data sequence D_n. Without loss of generality, we first focus on when there is single data value v_{n+1} is appended, and then generalize the procedure when values $\langle v_{n+1}, v_{n+2}, \ldots, v_m \rangle$ are appended in a batch.

When appending value v_{n+1} to D_n, it works like BIA (section 2.3.2.1) to get $\mathcal{SS}_{n+1}$ and local prominent streaks ended at n based on three different cases. Specifically, our maintenance algorithm aims to:

M1. identify streaks in case (1). Those streaks will not be in $\mathcal{SS}_{n+1}$ and they require no more isolation. So, we insert them into the R-tree;

M2. compute the updated set of situational streaks $\mathcal{SS}_{n+1}$ from case (2) and case (3). The streaks in $\mathcal{SS}_{n+1}$ will be stored explicitly, sorted by their values.

M3. resume minimality, as minimality may be violated at this point.

The reasons causing the minimality violation include:

(a) A perplexing streak $s \in \mathcal{P}_n$ may become a non-perplexing streak since D_{n+1}. For D_n, s can only prune streaks that it universally dominates through Lemma 5. But suppose that s further becomes a non-perplexing streak for D_{n+1}. Then, s is in $\mathcal{N}_{n+1}$ and can prune any streak that it dominates through Lemma 4. Recall that universally domination is harder to achieve than ordinary domination. When s goes from $\mathcal{P}_n$ to $\mathcal{N}_{n+1}$, it means now its 'pruning power' has increased. So now more streaks could be pruned by s and they should be removed accordingly in order to maintain minimality.

(b) A streak s, which is not universally dominated by any perplexing streak in $\mathcal{P}_n$, may now be *universally dominated* by a perplexing streak in $\mathcal{P}_{n+1}$, so that s should be removed. This happen when s was an analogous streak of some streak z in $\mathcal{SS}_n$ but z is no longer in $\mathcal{SS}_{n+1}$ due to case (1), or s is no longer the analogous streak of the extended z' in $\mathcal{SS}_{n+1}$ because of case (2) (i.e., $|z'|$ becomes longer and $|s| \geq |z'| \cdot \sigma$ does not hold anymore). Note that when s is not an analogous streak of any streak in $\mathcal{SS}_{n+1}$ and is

dominated by some streak in $\mathcal{P}_{n+1}$, s could be and should be removed in order to maintain minimality.

Algorithm 4 SOIA Maintenance Procedure (Only For Appending A Single Value)

```
1: procedure MAINTENANCE(v_{n+1})
2:     if n == 0 then
3:         P_1 = {(1, 1, v_1)};
4:         return;
5:     SS_{n+1} = ∅;
6:     for each streak (i, n, v) in SS_n do
7:         if v_{n+1} ≥ v then
8:             Insert (i, n + 1, v) to SS_{n+1};
9:         else if |n − i| ≥ σ then
10:            Insert (i, n, v) to Rtree;
11:    if no streak in SS_n has value v_{n+1} then
12:        if all streaks in SS_n have value < v_{n+1} then
13:            Insert (n + 1, n + 1, v_{n+1}) to SS_{n+1};
14:        else
15:            Select the streak (i, n, v) in SS_n whose value v > v_{n+1} and is the
    smallest;
16:            Extend it to be (i, n + 1, v_{n+1})
17:            Insert it into SS_{n+1};
18:    T = streaks in Rtree;
19:    Set P_n^T = T ∩ P_n;
20:    Find ΔN and P_{n+1} from P_n^T and SS_{n+1};
21:    for each streak y in ΔN do
22:        Remove streak s from Rtree if y ≻ s;
23:    for each streak p in P_{n+1} do
24:        Remove streak s from Rtree if s is universally dominated by p;
```

Algorithm 4 shows how the maintenance procedure works in the case of appending a single value.

Lines 2–4 are to initialize $\mathcal{P}_1$. Lines 6–17 work like BIA's maintenance, except that it needs to check the length of streak when inserting to the R-tree

(Line 9). Lines 18–24 are for resuming the minimality, corresponding to M3(a) and M3(b) discussed above. Specifically, we are meant to do two kinds of pruning:

- Pruning (i) Remove the streaks that are dominated by any streak in $skyline(\mathcal{N}_{n+1})$ (c.f. Lemma 4).
- Pruning (ii) Remove the streaks that are universally dominated by any streak in $\mathcal{P}_{n+1}$ (c.f. Lemma 5).

Let $\mathcal{T}$ be the streaks in the updated R-tree after executing Lines 2-14 of the maintenance procedure. So to carry out Pruning (i) and (ii) above, a basic method is to identify $\mathcal{N}_{n+1}$ by comparing each streak in $\mathcal{T}$ with the streaks in $\mathcal{SS}_{n+1}$, according to Definition 8. Once $\mathcal{N}_{n+1}$ is obtained, Pruning (i) above can be done. Next, identify streaks in $\{\mathcal{T} - N_{n+1}\} \cup \mathcal{SS}_{n+1}$ as $\mathcal{P}_{n+1}$, and use it to complete Pruning (ii).

For efficiency, we now show that Pruning (i) can be done by a much smaller subset $\Delta\mathcal{N} \subseteq \mathcal{N}_{n+1}$ instead, where $\Delta\mathcal{N} = \mathcal{N}_{n+1} - \mathcal{N}_n$. Especially, $\Delta\mathcal{N}$ can be obtained from $\mathcal{P}_n^{\mathcal{T}}$, where $\mathcal{P}_n^{\mathcal{T}}$ denotes $\mathcal{P}_n \cap \mathcal{T}$. Also, $\mathcal{P}_{n+1}$ can be identified in a more efficient way accordingly.

Lemma 8. *If a streak in $\mathcal{T}$ is dominated by any streak in $skyline(\mathcal{N}_{n+1})$, then such a streak is dominated by some streak in $\Delta\mathcal{N}$.*

Proof. We prove the lemma with the help of the following two lemmas,

Lemma 9. *For a streak $s \in \mathcal{T}$, if $s \notin \mathcal{P}_n^{\mathcal{T}}$, then $s \notin \mathcal{P}_{n+1}$.*

. Let s be a streak in the $\mathcal{T}$ but not in $\mathcal{P}_n$, i.e., $s \notin \mathcal{P}_n^{\mathcal{T}}$. Now, suppose to the contrary: $s \in \mathcal{P}_{n+1}$. This implies that there exists streak $z' = (i', j', v') \in \mathcal{SS}_{n+1}$, i.e., $j' = n + 1$, such that s overlaps with z', and $|s| \geq |z'| \cdot \sigma$. However, for $z' \in \mathcal{SS}_{n+1}$, there must be a situational streak $z = (i, j, v) \in \mathcal{SS}_n$ with the same starting position as z', i.e., $i = i'$ and $j = n$. It follows that s also overlaps with z, and $|s| \geq |z| \cdot \sigma$. Thus, $s \in \mathcal{P}_n^{\mathcal{T}}$, a contradiction. ■

Lemma 10.

$$skyline(\mathcal{N}_{n+1}) = skyline(skyline(\mathcal{N}_n) \cup \Delta\mathcal{N}).$$

. By Lemma 9, for any streak $s \in \mathcal{N}_n$, which means $s \notin \mathcal{P}_n^{\mathcal{T}}$, so that $s \notin \mathcal{P}_{n+1}$. Thus, by definition, such a streak s must be in $\mathcal{N}_{n+1}$. This implies $\mathcal{N}_n \subseteq \mathcal{N}_{n+1}$. So, we have $\mathcal{N}_{n+1} = \mathcal{N}_n \cup \Delta\mathcal{N}$. Further, if a streak s is in $\mathcal{N}_n$ but not in $skyline(\mathcal{N}_n)$, it is neither in $skyline(\mathcal{N}_{n+1})$, since the streak that dominates s is still in $\mathcal{N}_{n+1}$. That completes the proof since removing a non-skyline streak from a set would not affect the result of a skyline operation. ■

Lemma 10 implies that if a streak in $\mathcal{T}$ is dominated by $skyline(\mathcal{N}_{n+1})$, then it is dominated by $skyline(\mathcal{N}_n) \cup \Delta\mathcal{N}$. Furthermore, if a streak is dominated by $skyline(\mathcal{N}_n)$, it does not exist in $\mathcal{T}$ because: (1) $\mathcal{T}$ includes the streaks in $\mathcal{U}_n - \mathcal{SS}_n$, and the newly streaks ended at n by Line 8 of Algorithm 4. None of these newly added streaks are dominated by $skyline(\mathcal{N}_n)$; and (2) all streaks in $\mathcal{T}$ that are dominated by $skyline(\mathcal{N}_n)$ are pruned during any previous execution of the maintenance procedure. Thus, any streak in $\mathcal{T}$ that is dominated by

$skyline(\mathcal{N}_{n+1})$ must be dominated by some streak in $\Delta\mathcal{N}$. □

We proceed to discuss how to obtain $\Delta\mathcal{N}$ and $\mathcal{P}_{n+1}$. First, we state that $\Delta\mathcal{N}$ can be obtained from $\mathcal{P}_n^{\mathcal{T}}$.

Lemma 11. $\Delta\mathcal{N} \subseteq \mathcal{P}_n^{\mathcal{T}}$.

Proof. Any streak $s \in \mathcal{T}$ that s in $\Delta\mathcal{N} \subseteq \mathcal{N}_{n+1}$ cannot be in $\mathcal{P}_{n+1}$, so that its ending position is at most n. This implies $s \in \mathcal{N}_n \cup \mathcal{P}_n^{\mathcal{T}}$. By definition, $\Delta\mathcal{N} = \mathcal{N}_{n+1} - \mathcal{N}_n$. So, s cannot be in $\mathcal{N}_n$ but can only be in $\mathcal{P}_n^{\mathcal{T}}$, which gives $\Delta\mathcal{N} \subseteq \mathcal{P}_n^{\mathcal{T}}$. □

Lemma 12. $\Delta\mathcal{N} = \mathcal{N}_{n+1} \cap \mathcal{P}_n^{\mathcal{T}}$

Proof. Following Lemma 11, we have $\Delta\mathcal{N} \subseteq \mathcal{P}_n^{\mathcal{T}}$ and then

- $\mathcal{N}_n \cap \mathcal{P}_n^{\mathcal{T}} = \mathcal{N}_n \cap (\mathcal{P}_n \cap \mathcal{T}) = (\mathcal{N}_n \cap \mathcal{P}_n) \cap \mathcal{T} = \varnothing$
- $\mathcal{N}_{n+1} \cap \mathcal{P}_n^{\mathcal{T}} = (\mathcal{N}_n \cup \Delta\mathcal{N}) \cap \mathcal{P}_n^{\mathcal{T}} = (\mathcal{N}_n \cap \mathcal{P}_n^{\mathcal{T}}) \cup (\Delta\mathcal{N} \cap \mathcal{P}_n^{\mathcal{T}}) = \varnothing \cup \Delta\mathcal{N} = \Delta\mathcal{N}$

The lemma follows. □

Next, we state that $\mathcal{P}_{n+1}$ can be obtained from $\mathcal{P}_n^{\mathcal{T}}$, $\Delta\mathcal{N}$, and the updated $\mathcal{SS}_{n+1}$:

Lemma 13. $\mathcal{P}_{n+1} = (\mathcal{P}_n^{\mathcal{T}} - \Delta\mathcal{N}) \cup \mathcal{SS}_{n+1}$

Proof. We discuss $\mathcal{P}_{n+1}$ as two parts below,

(i) $\mathcal{P}_{n+1}$ includes $\mathcal{SS}_{n+1}$ by definition.

(ii) For streaks in $\mathcal{P}_{n+1} - \mathcal{SS}_{n+1}$, they are in $\mathcal{N}_n \cup \mathcal{P}_n^{\mathcal{T}}$. By Lemma 9, they must be in $\mathcal{P}_n^{\mathcal{T}}$; moreover, they cannot belong to $\mathcal{N}_{n+1}$. That is, they should be in $\mathcal{P}_n^{\mathcal{T}} - \mathcal{P}_n^{\mathcal{T}} \cap \mathcal{N}_{n+1}$, where $\mathcal{P}_n^{\mathcal{T}} \cap \mathcal{N}_{n+1} = \Delta\mathcal{N}$ by Lemma 12.

(i) gives $\mathcal{SS}_{n+1}$ and (ii) gives $\mathcal{P}_n^{\mathcal{T}} - \Delta\mathcal{N}$. The lemma thus follows. □

With Lemmas 11, 12 and 13, we shall obtain $\Delta\mathcal{N}$ and $\mathcal{P}_{n+1}$ as follows:

1. For each streak p in $\mathcal{P}_n^{\mathcal{T}}$, compare p with all streaks in $\mathcal{SS}_{n+1}$.

   If there does not exist any streak z in $\mathcal{SS}_{n+1}$ such that p overlaps with z and $|p| \geq |z| \cdot \sigma$, then p belongs to $\mathcal{N}_{n+1}$. For efficiency sake, we insert p to $\Delta\mathcal{N}$ (c.f. Lemma 8). Else, p belongs to $\mathcal{P}_{n+1}$ and we insert it into $\mathcal{P}_{n+1}$.

2. Expand $\mathcal{P}_{n+1}$ by including all streaks in $\mathcal{SS}_{n+1}$.

The above gives the details of Line 20 in Algorithm 4. The remaining Lines 21–24 carry out the pruning based on the discussed lemmas so far.

Example 7. *Let us revisit the example of appending $v_{20} = 5$ as in Figure 2.3, with $\sigma = 0.75$. Recall that before the value is appended as in Figure 2.2, we have:*

- $\mathcal{SS}_{19} = \{s_1, s_2, s_3, s_4\}$
- $\mathcal{P}_{19} = \{s_1, s_2, s_3, s_4, s_5\}$
- $\mathcal{N}_{19} = \{s_6, s_7, s_8, s_9, s_{10}, s_{11}\}$
- *Rtree* $= \{s_5, s_6, s_7, s_{11}\}$ *because* s_9 *and* s_{10} *are not in skyline*$(\mathcal{N}_{19})$ *and* s_8 *is universally dominated by* s_5.

Appending v_{20} requires us to compute the new $\mathcal{SS}_{20}$ and local prominent streaks ended at 19. Running Lines 6–18 will result in:

- $\mathcal{SS}_{20} = \{s'_3(8, 20, 4), s'_4(1, 20, 1), s_{new}(14, 20, 5)\}$, *as explained in Example 2.*
- $\mathcal{T} = \{s_1, s_2, s_5, s_6, s_7, s_{11}\}$, *as s_1 and s_2 get inserted into the R-tree according to case M1 (Line 10 in Algorithm 4).*

Lines 19–20 then identify the following:

- $\mathcal{P}_{19}^{\mathcal{T}} = \mathcal{T} \cap \mathcal{P}_{19} = \{s_1, s_2, s_5\}$.
- $\Delta\mathcal{N} = \{s_5\}$ *because in $\mathcal{P}_{19}^{\mathcal{T}}$, only s_5 has length less than $|z| \cdot \sigma$, where z denotes any streak in $\mathcal{SS}_{20}$.*
- $\mathcal{P}_{20} = (\mathcal{P}_{19}^{\mathcal{T}} - \Delta\mathcal{N}) \cup \mathcal{SS}_{20} = \{s_1, s_2, s_{new}, s'_3, s'_4\}$.

Lines 21–22 remove s_7 from the R-tree because $s_5 \in \Delta\mathcal{N}$ and $s_5 \succ s_7$. Next, Lines 23–24 check if any streak in the R-tree is universally dominated by streaks in $\mathcal{P}_{20}$ and prune them accordingly. In the example, no such streaks are found, and the minimality of the streaks stored in R-tree is maintained.

Now we generalize the discussions above to handle the case that a batch of data values $\langle v_{n+1}, v_{n+2}, \ldots, v_m \rangle$ is appended to D_n that results in D_m. Algorithm 5 shows the pseudocode of this maintenance procedure, and we have:

- Upon each value $v_{n+k} \in \{v_{n+1}, v_{n+2}, \cdots, v_m\}$, Lines 3–23 work similar to Lines 2–20 in Algorithm 4 that it computes corresponding $\mathcal{P}_{n+k}$ and $\Delta\mathcal{N}$

respectively. $\mathscr{N}$ is used to collect the streaks in $\Delta\mathcal{N}$ generated in Line 22 for each value v_{n+k}, so that in the end we have $\mathscr{N} = \mathcal{N}_m - \mathcal{N}_n$.

- Similar to Algorithm 2, we store the new local prominent streaks in a Buffer B (Line 12) and then insert them into the R-tree in a batch (Line 24) to improve the I/O efficiency.

- Lines 25–28 are to prune the R-tree to guarantee its space optimal properly, according to Lemma 8 and Lemma 5.

2.3.3.2 SOIA LOOKUP

The LOOKUP procedure of SOIA is the same as BIA's LOOKUP procedure. Algorithm 6 shows the pseudo-code when dealing with a new parameter σ' (with $\sigma' \geq \sigma$) and a new parameter k' from a user. It first obtains the top-k' situational streaks by a linear scan of $\mathcal{SS}_n$, since streaks in $\mathcal{SS}_n$ are sorted by values. Then, for each such situational streak z, it constructs a query using the following region:

$$Q = [|z| \cdot \sigma', +\infty] \ \times \ [0, z.i) \ \times \ [z.v \cdot \sigma', +\infty]$$

That region contains exactly the set $\mathcal{AS}(z)$ of analogous streaks of z. We can thus apply the BBS skyline algorithm in [49] to compute $\mathcal{HM}(z)$ (the constrained skyline) from the R-tree.

Example 8. *Following Example 7, given $\sigma = 0.6$ and $k = 2$, now we have*

- $Rtree = \{s_1, s_2, s_5, s_6, s_{11}\}$

Algorithm 5 SOIA Maintenance Procedure

1: **procedure** MAINTENANCE($\langle v_{n+1}, v_{n+2}, \ldots, v_m \rangle$)
2: $\mathscr{N} = \varnothing$;
3: **for** $k = 1$ to $m - n$ **do**
4: **if** $n == 0$ and $k == 1$ **then**
5: $\mathcal{P}_1 = \{(1, 1, v_1)\}$;
6: **continue**;
7: $\mathcal{SS}_{n+k} = \varnothing$;
8: **for each** streak $(i, n + k - 1, v)$ in $\mathcal{SS}_{n+k-1}$ **do**
9: **if** $v_{n+k} \geq v$ **then**
10: Insert $(i, n + k, v)$ to $\mathcal{SS}_{n+k}$;
11: **else if** $|n + k - 1 - i| \geq \sigma$ **then**
12: Insert $(i, n + k - 1, v)$ to Buffer B;
13: **if** no streak in $\mathcal{SS}_{n+k-1}$ has value v_{n+k} **then**
14: **if** all streaks in $\mathcal{SS}_{n+k-1}$ have value $< v_{n+k}$ **then**
15: Insert $(n + k, n + k, v_{n+k})$ to $\mathcal{SS}_{n+k}$;
16: **else**
17: Select the streak $(i, n + k - 1, v)$ in $\mathcal{SS}_{n+k-1}$ whose value $v > v_{n+k}$ and is the smallest;
18: Extend it to be $(i, n + k, v_{n+k})$
19: Insert it into $\mathcal{SS}_{n+k}$;
20: $\mathcal{T}$ = streaks in Rtree $\cup$ B;
21: Set $\mathcal{P}^{\mathcal{T}}_{n+k-1} = \mathcal{T} \cap \mathcal{P}_{n+k-1}$;
22: Find $\Delta\mathcal{N}$ and $\mathcal{P}_{n+k}$ from $\mathcal{P}^{\mathcal{T}}_{n+k-1}$ and $\mathcal{SS}_{n+k}$;
23: $\mathscr{N} = \mathscr{N} \cup \Delta\mathcal{N}$;
24: Insert streaks in B into Rtree;
25: **for each** streak y in $\mathscr{N}$ **do**
26: Remove streak s from Rtree if $y \succ s$;
27: **for each** streak p in $\mathcal{P}_m$ **do**
28: Remove streak s from Rtree if s is universally dominated by p;

Algorithm 6 SOIA Lookup Procedure

```
1: procedure LOOKUP(σ', k')
2:     Z = Get-Top-SS(SS_n, k');
3:     for each streak z in Z do
4:         Q = [|z| · σ', +∞] × [0, z.i) × [z.v · σ', +∞];
5:         HM(z) = BBS(Rtree, Q); // compute constrained skyline
```

- $\mathcal{SS}_{19} = \{s'_3, s'_4, s_{new}\}$.

When querying with parameter $\sigma' = 0.6$ *and* $k' = 2$*: for* s_{new}*, find out its analogous streaks in Rtree that* $\mathcal{AS}(s_{new}) = \{s_{11}\}$*, and its historic moment is* $skyline(\mathcal{AS}(s_{new})) = \{s_{11}\}$*; for* s'_3*, there is no analogous streak for it. So now the historic moment is* $\{s_{11}\}$.

2.4 Case Study

2.4.1 Microsoft's stock price

The first sequence dataset is Microsoft (NASDAQ:MSFT) daily stock price[10] from year 1986 to year 2014. We set the similarity threshold σ as 1 and monitor the historic moments of the top-1 situational streak. On 11 June 2014 we got the following top-1 situational streak:

(2014–6–05, 2014–6–11, 40.35)

and its corresponding historic moments are:

(1999–12–16, 2000–1–05, 40.36)

[10] `http://finance.yahoo.com/q/hp?s=MSFT`

(1999–12–21, 2000–1–03, 41.52)

(1999–12–22, 2000–1–03, 41.77)

(1999–12–22,1999–12–31, 41.83)

(1999–12–22,1999–12–30, 42.08)

(1999–12–16, 2000–1–03, 40.40)

It turns out that all those historic moments *happened around the end of 1999*, meaning that Microsoft had not had such a long streak of high stock prices for almost 14 years. Indeed, a real news[11] was reported on 11 June 2014 based on the above:

> *Microsoft stock inching closer to all-time high. Don't look now, but Microsoft (NASDAQ:MSFT) stock is at a 14-year high and is approaching its all-time high reached just before the dot-com crash.*

2.4.2 Beijing's temperature

The second sequence dataset is the average daily temperature of Beijing[12] from year 1995 to year 2014. We set the similarity threshold σ as 1 and monitor the historic moments of the top-1 situational streak. On 29 July 2010 we got the following top-1 situational streak:

(2010–7–27, 2010–7–29, 85.5)

and its corresponding historic moments are:

[11] `http://www.scalper1.com/microsoft-stock-inching-closer-to-all-time-high.html`
[12] `http://academic.udayton.edu/kissock/http/Weather/gsod95-current/CIBIEJNG.txt`

(2000–7–03, 2010–7–06, 86.8)

(2000–7–11, 2000–7–14, 87.6)

(2000–7–22, 2000–7–26, 85.6)

(1999–7–23, 1999–7–29, 87.4)

These historic moments imply that the last time that Beijing had such a long streak of high temperature was almost 10 years ago, and this observation was reported in the news on 29 July 2010:[13]

> *High temperature days in Beijing July last longest in the past ten years. [...]*

2.4.3 Taiwan seismic datasets

The third dataset is the ground motion sensor stream of Taiwan[14]. Ground motion is the movement of the earth's surface. Seismologists can utilize ground motion data to study and even predict some geological activities such as earthquake [5]. The datasets are streams of sample *counts*[15] for every 50ms, since year 1998, with one data sequence per monitoring station. Each data sequence is about 350GB in size and BIA required to build an index of size 500GB, exceeding the disk size of our experimental platform (detailed configuration in section 2.5). In contrast, SOIA only required 2GB to house the index, making this case study feasible.

[13] `http://news.163.com/10/0729/08/6COD18AV000146BC.html`

[14] `http://ds.iris.edu/ds/nodes/dmc/data/`

[15] *Count* is a scale unit of ground motion. A "count" value of 3.27508e9 indicates ground motion of 1 meter/second.

We set the similarity threshold σ as 0.9 and monitor the historic moments of the top-10 situational streaks. We got a number of interesting findings there when we study the sequence data from one such station, namely station "KMNB". Specifically, it illustrates that while historic moments that happened long ago are useful, those happened very recently are also helpful to highlight the importance of a prominent streak in certain domains.

For example, we got a situational streak s_a on 3 Feb 2016. That was a streak of length of 202 units (i.e., 10.1s) with 40334 counts. Let's look at its corresponding historic moments:

(2016-01-30-03:43, length=184, count=40391)
(2016-01-30-03:43, length=185, count=40188)
(2016-01-30-03:42, length=189, count=39161)

We can see that its historic moments just happened four days before 3 Feb 2016. That implied it was not a singular prominent event and worth paying attention. In fact, based on our record [63], an earthquake happened 2 days later, on 5 Feb 2016. As another example, we got another situational streak s_b on 27 Feb 2010 of length 238 units (i.e., 11.9s) with 42492 counts. Its historic moments are:

(2010-02-26-01:49, length=219, count=40949)
(2010-02-26-01:48, length=223, count=40232)
(2010-02-26-01:48, length=230, count=40119)

It is vigilant that its historic moments just happened one day before. Actu-

ally, an earthquake happened 5 days later, on 4 March 2010, according to [63].

How can one tell the above historic moments are fair indicator of earthquake but not normal energy release? In order to answer that question, we ran SOIA on the dataset and obtain the following statistics:

> *Among all (situational) streaks of length $>$ 200 and value $>$ 40000 (e.g., s_a and s_b above), their corresponding historic moments happened 23 days ago, on average.*

One plausible message of the above is that — historic moments that are more recent to the situational streaks (and farther away than the mean, 23 days), the more significant they are as an earthquake indicator. To cross check that claim, we found that for all situational streaks of length $>$ 200 and value $>$ 40000 in the dataset, whenever their corresponding historic moments happened before 23+ days, no earthquake was reported within weeks after those situational streaks happened.

2.5 Performance Study

In this section, we evaluate the performance of SOIA using five real sequence datasets and compare with BIA. The experimental platform has a 2.8 GHz Intel i5 CPU, 8GB RAM and 256GB hard disk. The program is implemented in C on MacOS. Table 2.2 lists the information about the datasets, which are ordered based on their data sizes. Datasets D5 was too large that BIA would require index space larger than our disk, therefore, we only used a fraction of D5 in

order to make BIA runnable in this section.

2.5.1 Overall Comparison

We first look at the overall performance of BIA and SOIA when the full dataset is available. We evaluate the performance of BIA and SOIA in terms of their (a) index building time (maintenance time), (b) query time (look up time), and (c) index space.

We use $\sigma = 0.5$ in building the index structure (MAINTENANCE procedure). We create five user lookup workloads: W_1, W_2, W_3, W_4, W_5, where a workload W_i mimics a user exploring the dataset by trying out $10 \cdot i$ different k and σ' historic moments look up. Therefore, W_1 consists of 10 lookups (e.g., a casual user) and W_5 consists of 50 lookups (e.g., a serious journalist). When reporting the look up time, we report the average running time of W_1 to W_5. That represents the total wait time for a user in one interactive session. In the experiment, each lookup

[16] https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption
[17] http://alumni.cs.ucr.edu/mueen/OnlineMotif/index.html
[18] http://ita.ee.lbl.gov/html/contrib/WorldCup.html
[19] http://ds.iris.edu/ds/nodes/dmc/forms/breqfast-request/

Table 2.2: Summary of Datasets

Dataset	Size(MB)	Length	Description
D1	14.1	1074637	household global minute-averaged current intensity from Dec 2006 to Dec 2008[16]
D2	21.3	1600237	household global minute-averaged active power from Dec 2006 to Dec 2009
D3	27.5	1802000	EEG time series datasets [17]
D4	32.5	2764800	number of requests to World Cup 98 website per second from April to May 1998[18]
D5	52.4	3456000	Taiwan KMNB station ground motion from 1 Jan 2017 to 2 Jan 2017 [19]

query randomly choses a value between 1 and 10 for k and randomly chose a value between σ and 1 as σ'.

Figure 2.4 shows the results. Since SOIA has to invest some time to identify and prune redundant streaks in order to maintain a minimal space index for latter use, its one-off maintenance time is higher than BIA (Figure 2.4a). Nevertheless, we see that such investment is worthwhile because the lookup time of SOIA is 9.58× (D1) to 184.14× (D4) better than BIA (Figure 2.4b), which gives users much shorter waiting time during the exploration. Figure 2.4c shows the space

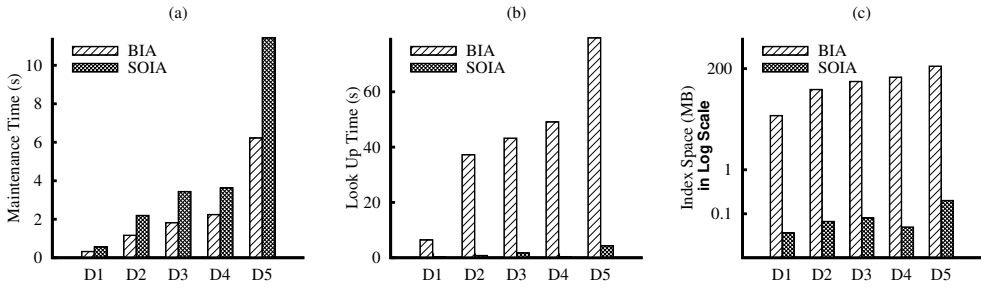


Figure 2.4: BIA vs. SOIA

requirement of SOIA is much smaller than that of BIA. On D5 (a largely trimmed version of Taiwan's ground motion data), monitoring historic moments for just one station already requires BIA to build an index bigger than 200MB.

The minimal index size of SOIA is also the key factor that leads to its excellent historic moment lookup performance (other factors include the size and the value of the streak, etc; see Algorithm 6). Compared with SOIA, BIA takes 656× (D1) to 2898× (D3) more index space. Table 2.3 lists the index space as well as the number of streaks stored using BIA and SOIA respectively. For SOIA, the number of streaks in $Skyline(\mathcal{N}_n)$ and $\mathcal{P}_n$ are also reported. In fact,

the sizes of the index structures are proportional to the numbers of streaks in $\mathcal{LPS}_n$. In the following sections, we only include the index size in our discussions.

Table 2.3: Number of Streaks Maintained by BIA and SOIA

Method	BIA		SOIA			
Dataset	Index Space (MB)	Number of Streaks Stored	Index Space (MB)	Number of Streaks Stored	Number of Streaks in $Skyline(\mathcal{N}_n)$	Number of Streaks in $\mathcal{P}_n$
D1	17.06	299868	0.026	434	433	8
D2	66.91	1175934	0.061	1012	987	31
D3	127.55	2241976	0.044	676	662	14
D4	101.87	1790601	0.080	1362	1337	25
D5	226.75	3985692	0.199	3291	3289	20

2.5.2 Historic Moment Exploration with Data Update

Next, we look at the performance of SOIA and BIA for maintaining the index structure online. That is, whenever a value arrives, BIA inserts new local prominent streaks into the R-tree immediately, and SOIA maintains its space optimal property by inserting and pruning the R-tree immediately.

In this experiment, we regard the first 98% of a dataset as the initial dataset and its index structure has been built by MAINTENANCE procedure already. Then, we examine the performance of SOIA and BIA regarding a data append of the last x% of the dataset, where $x = 0.1$, 0.5, 1, and 2.

Figures 2.5 and 2.6 show the experiment results. In the figures, we report:

(a) the time of the MAINTENANCE procedure in order to handle data appending.

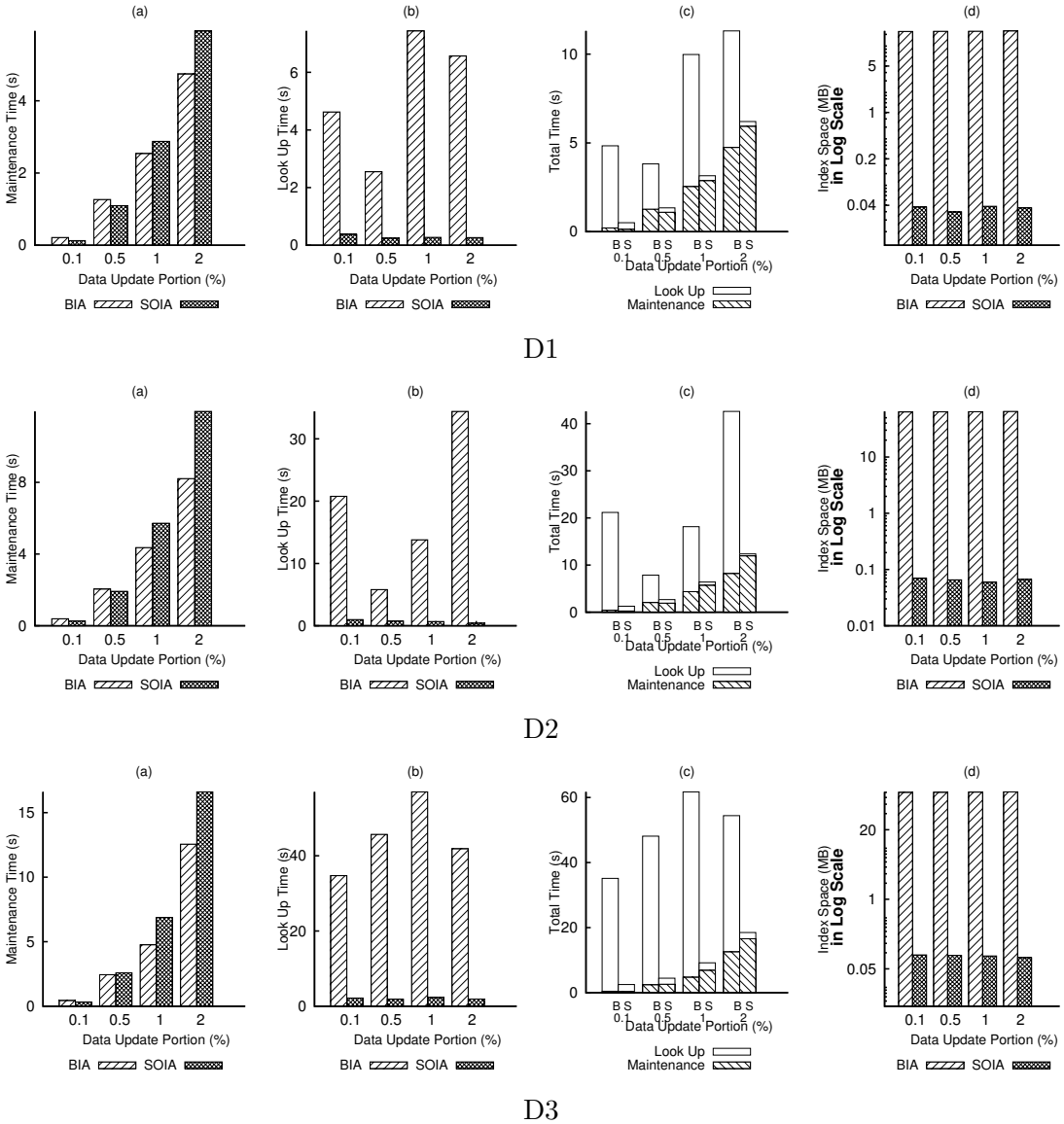
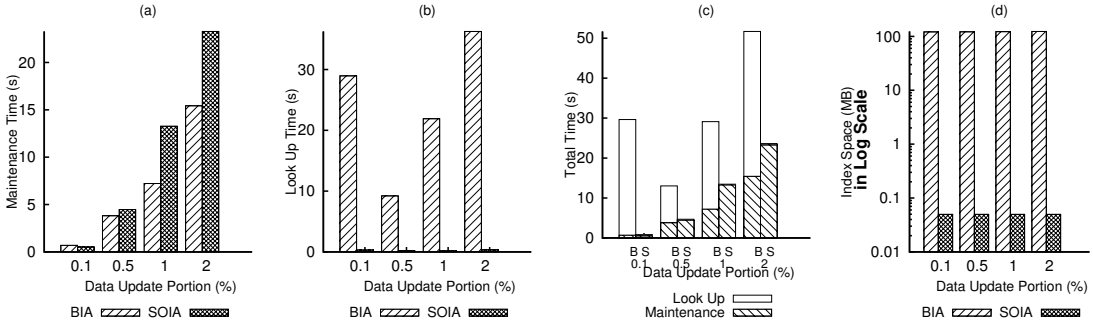
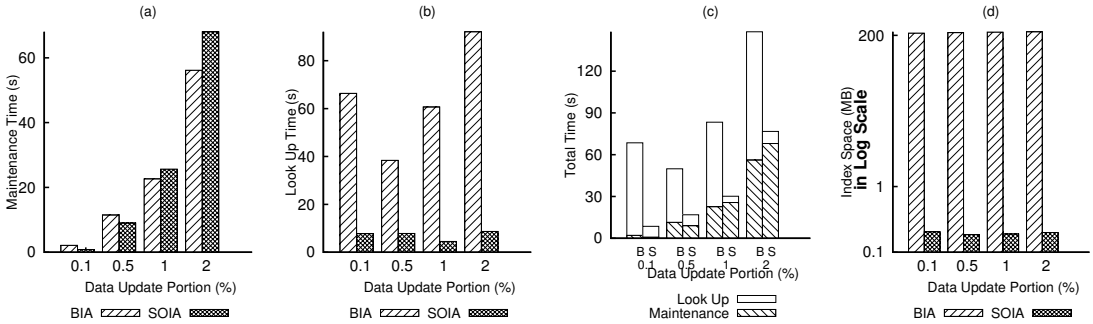


Figure 2.5: BIA vs. SOIA under data update (D1, D2, D3)



D4



D5

Figure 2.6: BIA vs. SOIA under data update (D4, D5)

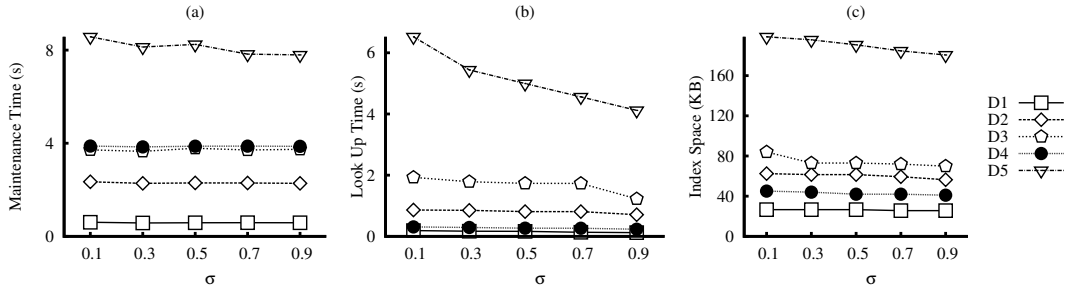


Figure 2.7: Varying σ

(b) the average time of running time of W_1 to W_5.

(c) the total of (a) and (b).

(d) the size of the index after a data update.

The maintenance times of SOIA and BIA are similar (see sub-figures (a)) because the advantage of having a smaller index in SOIA is offset by the extra effort spent on maintaining minimality. Nevertheless, SOIA is superior in terms of lookup performance (see sub-figures (b)). So, consider the time from getting the new data to the time that a user finishes a particular session of historic moment exploratory (see sub-figures (c)), SOIA is much better than BIA. With no surprise, SOIA maintains a smaller index size all the way (see sub-figures (d)).

2.5.3 Sensitivity Study

Here, we look at the impact of parameters σ and k on the performance of SOIA.

We first look at the influence of σ. In this experiment, given the full dataset is available, we try different values for σ: 0.1, 0.3. 0.5. 0.7, 0.9. Figure 2.7a

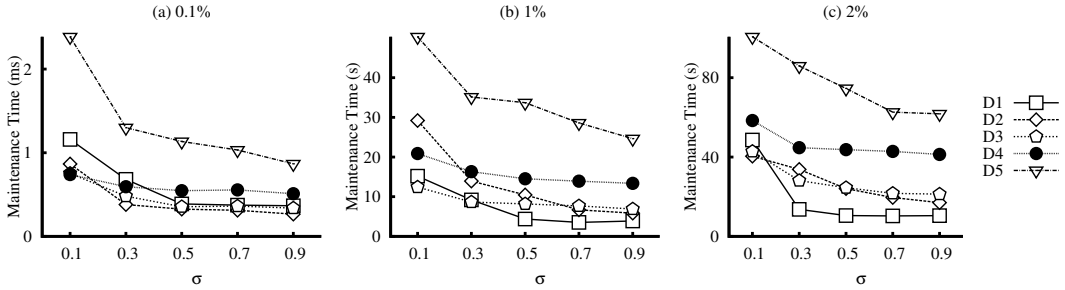


Figure 2.8: Varying σ When Updating

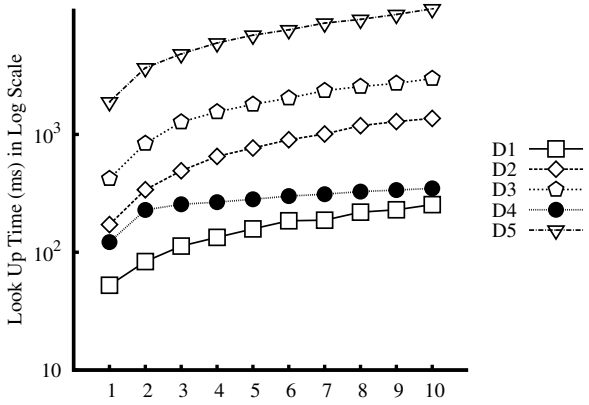


Figure 2.9: Varying k

shows that different σ values do not influence the maintenance time of SOIA much. That is because the maintenance of SOIA is dominated by the time of scanning all $\mathcal{LPS}_n$ to get $\mathcal{P}_n$, which is independent of σ there. Figure 2.7b shows that a higher σ value during maintenance would make historic moment lookup more efficient because that would result in a smaller index as shown in Figure 2.7c.

Figure 2.8 shows the time for maintaining space optimal online when data updates. In that experiment, we report the results the maintenance time when the remaining 0.1%, 1% and 2% of data are inserted, in case that the index has

been built for 98% of the original data already. When σ increases, the maintenance procedure takes less time. That is because a higher σ value would reduce the number of perplexing streaks, and for each perplexing streak p, the maintenance algorithm is required to remove streaks that are universally dominated by p from the R-tree (Algorithm 4 Lines 23–24). As a result, the maintenance time decreases when σ increases.

Lastly, we look at the impact of k on the performance of SOIA. Note that k has no impact on maintenance phase of SOIA, so we only report the average execution time of the lookup phase based on the five workloads, with $\sigma = 0.5$.

Figure 2.9 shows that the lookup time generally increases with the value k. That is because the lookup procedure looks for the historic moment for each top-k situational streak. Increasing k would then increase the number of skyline computational calls to the index.

2.6 Summary

In this chapter, we present SOIA, an efficient algorithm to find out "historic moments" over large data sequence. The algorithm is both space-optimal and incremental, which can be applied in online applications such as data monitoring. Also, we show that, compared with singular "prominent streaks", our "historic moments" can give more insightful stories over the data.

For future works, we may study how to design the algorithm when defining "historic moments" to be more general, e.g., similarity threshold σ is given differently on streak value and length. We may also study, apart from R-tree, a

kind of index structure to accelerate the computing more efficiently.

Chapter 3

Range Counting over Data Stream

Previous chapter studies finding interesting facts over data sequence of large volume. In this chapter, we study the the problem incurred by data sequence of large velocity. We present LSH-sketch, a sketch designed for range-count queries over rapid data stream. The remainder of this chapter is organized as follows: Section 3.1 gives the preliminary and background. Section 3.2 presents LSH-Sketch in detail. Section 3.3 presents the empirical results. Section 3.4 discusses the related work.

3.1 Preliminary and Background

In this section, we first put down the problem definition (section 3.1.1). Next, we give some background on sketches (section 3.1.2). Finally, we out-

line the concept of locality sensitive hashing (LSH) for understanding this work (section 3.1.3).

3.1.1 Problem Definition

Given a data stream $S = (v_1, v_2, v_3, \cdots, v_n)$ that has n values so far, each value v_i in S is from the domain $[0, D)$, we aim to build a range-count sketch by reading each value $v \in S$ only once. For each value insertion, we expect $O(1)$ time complexity. For a range-count query $Q[v_L, v_R]$ that is posed over S, the approximate query answer, $\hat{f}$ of Q, has a probability of at least $1 - \delta$ to be bounded by:

$$f \leq \hat{f} \leq f + \Delta$$

where δ is a parameter derived from the space budget, f is the exact query answer, Δ is the expected (one-sided) error.[1] Concerning the streaming model, both *cash register* and *strict turnstile* models [26] are supported.

3.1.2 Sketch

There are sketches to support point-count queries [16] and heavy-hitter queries [7]. Heavy-hitter queries aim to provide more accurate count estimates for values with top k highest counts, while there is no accuracy bound for those remaining values [45]. Therefore, in the context of range-counting with probabilistic guarantee, point-count sketches are the baselines.

[1] Usually $\Delta = O(n)$ [17].

There are two approaches to use point-count sketches to answer range-count queries. The first approach is to degenerate a range-count query $Q[v_L, v_R]$ into $(v_R - v_L + 1)$ point-count queries and aggregate their approximated counts from a point-count sketch (section 3.1.2.1). The second approach is to degenerate the domain into regular intervals and treat each interval as a point and insert them into a point-count sketch (section 3.1.2.2). The first approach has large error. The second approach needs to maintain multiple sketches, each for different interval size, which violates the $O(1)$ insertion latency requirement and makes the insertion throughput unacceptable.

3.1.2.1 Range-Counting using Point-Count Sketch

They are many point-count sketches in the literature [23, 51, 52, 66], but most of them are optimizations or variants of Count-Min Sketch [16] (CM-Sketch for short). All these sketches share the same theoretical error bound but with different empirical performance. Our LSH-Sketch can play the same role as CM-Sketch, where various optimizations developed on top of CM-Sketch technically can be applied to LSH-Sketch as well. Therefore, in this work, we put our focus on CM-Sketch and leave the study of its other optimizations as our future work.

A CM-sketch is composed of a table of *cells* of depth d and width w. Each cell has a counter. The counters are maintained through d *pairwise independent* hash functions. For example, each function U_i is chosen from a universal hashing family that maps a value v to a value in $\{1, \ldots, w\}$. Figure 3.1 shows an example of a CM-Sketch with $d = 4$ rows and $w = 7$ columns. Let $CM[x, y]$ be the cell counter under row x and column y. When inserting an element v into the sketch,

$d = 4$ counters $CM[i, U_i(v)]$ (i =1 to 4) would get incremented by 1. When answering a query q_v about the count of value v from the sketch, the value $\min_i\{CM[i, U_i(v)]\}$ is returned.

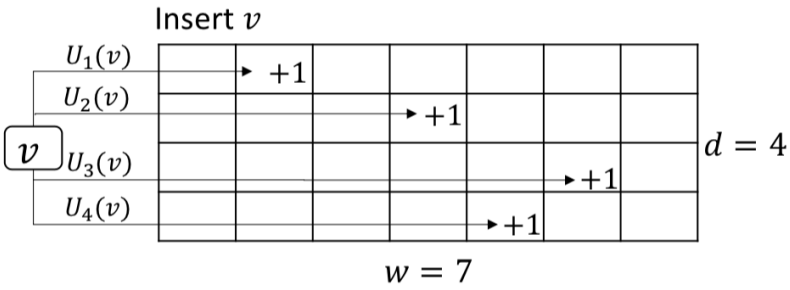


Figure 3.1: CM-Sketch Structure

By fixing two parameters δ and ε, and defining depth $d = \lceil \ln(\frac{1}{\delta}) \rceil$ and width $w = \lceil \frac{e}{\varepsilon} \rceil$ according to the parameters (where e is the base of natural logarithm), it has been shown in [16] that for CM-Sketch, the returned answer $\hat{f}_v$ of a point-count query q_v has a probability at least $1 - \delta$ to be bounded by:

$$f_v \leq \hat{f}_v \leq f_v + \varepsilon n$$

where f_v is the exact query answer. Inserting an item into CM-Sketch requires updating d counters, so its insertion latency is $O(1)$ because d is a user-given constant.

There has been no formal discussion of using a point-count sketch like CM-Sketch to serve as a baseline for answering range-count queries. Nonetheless, it is straightforward that we can degenerate a range-count query $Q[v_L, v_R]$ into $(v_R - v_L + 1)$ point-count queries $q_{v_L}, \ldots, q_{v_R}$ and sum up their approximate counts $\hat{f}_{v_L} + \ldots + \hat{f}_{v_R}$ obtained from a CM-sketch as an approximate answer $\hat{f}_Q$.

However, such a $\hat{f}_Q$ will have a poor error bound. Specifically, let $\hat{f}_Q = \sum \hat{f}_{v_i}$ where $v_i \in [v_L \dots v_R]$, $|Q| = (v_R - v_L + 1)$, and f_Q is the exact answer to Q; then

Corollary 3. *Using CM-Sketch, there is a probability of at least* $1 - \delta$ *that the result* $\hat{f}_Q$ *of a range-count query* Q *is bounded by:*

$$f_Q \leq \hat{f}_Q \leq f_Q + |Q|\varepsilon n. \tag{3.1}$$

Proof. Let X denote the error $\hat{f}_Q - f_Q$. We first consider the case when $d = 1$. It is known that for CM-Sketch [16], the expected error $\mathbf{E}(X)$ for a point-query is at most $\frac{n}{w}$. As $w = \lceil \frac{\mathrm{e}}{\varepsilon} \rceil$ and there are $|Q|$ points in the range-count query Q, by linearity of expectation, we have

$$\mathbf{E}(X) \leq \frac{|Q|\varepsilon n}{\mathrm{e}}.$$

As X is a nonnegative random variable, we can apply Markov's inequality and get

$$\mathbf{Pr}(X > |Q|\varepsilon n) \leq \frac{\mathbf{E}(X)}{|Q|\varepsilon n} \leq \frac{1}{\mathrm{e}}.$$

As by definition $\hat{f}_Q = f_Q + X$, we have

$$\mathbf{Pr}(\hat{f}_Q > f_Q + |Q|\varepsilon n) \leq \mathrm{e}^{-1}$$

when $d = 1$ for CM-sketch.

When $d > 1$, the probability that the minimum among all d estimates has error exceeding $|Q|\varepsilon n$ is equal to the probability that every row has error exceeding $|Q|\varepsilon n$. As the d hash functions are pairwise independent, we have

$$\mathbf{Pr}(\hat{f}_Q > f_Q + |Q|\varepsilon n) \leq \mathrm{e}^{-d}.$$

Since $\mathrm{e}^{-d} \leq \delta$, we have

$$\mathbf{Pr}(\hat{f}_Q \leq f_Q + |Q|\varepsilon n) \geq 1 - \delta.$$

□

For a range query of size $|Q|$, using the above baseline approach requires $|Q|$ bucket accesses per row. Therefore, the query latency is $d|Q|$, which is $O(|Q|)$ as d is a constant.

3.1.2.2 Range Counting using Multi-Level Sketch

To support range-count query more efficiently, there is an extension in [16] that partitions the domain $[0, D)$ under different granularities and maintains a CM-Sketch for each granularity. For example, we can maintain another CM-Sketch whose "points" are essentially representing intervals of size 2: $[0, 2), [2, 4), \ldots, [D-2, D)$. It was suggested to maintain $\log D$ levels (of different interval sizes) of CM-Sketches [16] since the query range would not always align with the partition boundaries. Therefore, the insertion of each item requires $d \log D$ cell accesses, which yields a poor insertion latency: $O(\log D)$. For a

range count query Q, this approach partitions Q into $2\log|Q|$ intervals at most, and each interval accesses the corresponding sketch for its counts. Therefore, this approach accesses $2\log|Q|$ sketches, where each requires d bucket accesses, resulting in a query latency of $O(\log|Q|)$.

Corollary 4 ([16])**.** *Using Multi-Level Sketch, there is a probability of at least* $1-\delta$ *that the result* $\hat{f}_Q$ *of a range-count query* Q *is bounded by:*

$$f_Q \leq \hat{f}_Q \leq f_Q + 2\varepsilon n \log D \log|Q|. \tag{3.2}$$

3.1.3 Locality Sensitive Hashing (LSH)

Locality sensitive hashing (LSH) has been an important (approximation) technique for a variety of problems including nearest-neighbour (NN) search and clustering [21, 27, 59]. The principle behind LSH is that it hashes input items so that similar items map to the same buckets with high probability.

An LSH family $\mathcal{F}$ is a set of functions $h : \mathcal{M} \to \mathcal{S}$ that each maps elements from a metric space $\mathcal{M}$ to a bucket $s \in \mathcal{S}$. For any two points $x, y \in \mathcal{M}$, let $dist(x, y)$ be the distance between them. For a function h chosen uniformly at random from $\mathcal{F}$, there exist r_1 and r_2 with $r_1 < r_2$ such that h satisfies the following conditions:

- if $dist(x, y) \leq r_1$, then $h(x) = h(y)$ with probability at least p_1;
- if $dist(x, y) \geq r_2$, then $h(x) = h(y)$ with probability at most p_2;
- $p_1 > p_2$.

Families that satisfy the above are called (r_1, r_2, p_1, p_2)-sensitive. When supporting (approximate) nearest neighbour search in a high-dimensional space, the following implementation of (r_1, r_2, p_1, p_2)-sensitive hash family is commonly adopted [21, 27, 50, 59]:

$$h(\vec{x}) = \left\lfloor \frac{\vec{a} \cdot \vec{x} + b}{r} \right\rfloor \tag{3.3}$$

where

1. $\vec{a}$ is a random vector with each $a_i \sim N(0, 1)$;
2. b is random variable sampled from uniform distribution: $b \sim U(0, r)$, and is an offset;
3. r is the quantization step chosen according to the data [50], and it is manually set, e.g., $r = 32$.

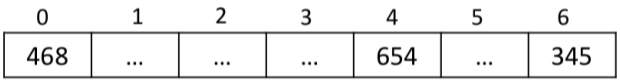
$\vec{a}$ is drawn from a normal distribution because normal distribution is a p-stable distribution, which is necessary to preserve the locality when reducing the dimensions from a higher dimensional space.

3.2 LSH-Sketch

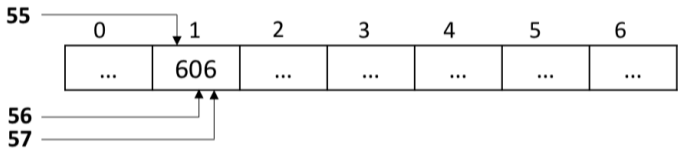
Conventional sketches deliberately use hash functions that minimize collisions. Consider CM-sketch with $d = 1$ and $w = 7$ in Figure 3.2(a). Assume its random hash function is h and $h(55) = 4$, $h(56) = 6$, and $h(57) = 0$. As collisions could happen, it is possible that the count in a cell is contributed by

multiple values. Since the exact number of values that hash into the same cell is unknown, there is no way to adjust any over-counting. That problem affects the accuracy of range-counting. For example, if a query range is 55 to 57, CM-sketch returns an approximate count of $654 + 345 + 468$.

LSH-sketch in contrast can tell certain information to adjust over-counting. Specifically, consider the same sketch structure as in Figure 3.2(a) but using LSH as the hash function. Since the values 55, 56, 57 are close to each other in the domain, they could be hashed to the same cell, e.g., $h(55) = h(56) = h(57) = 1$. Although the exact number of values that hash into the same cell is still unknown, we at least know that we shall not sum the count from that cell three times. Specifically, if the query range is 55 to 57, we know that we should not return $606 + 606 + 606$, which over-counts the same cell thrice. Instead, we know the estimated count of that query is at most 606.



(a) Universal hashing cannot adjust over-counting



(b) Locality Sensitive Hashing can adjust over-counting

Figure 3.2: Ability to adjust over-counting

Our LSH-sketch exactly follows this intuition. LSH-sketch is structural-wise

the same as CM-Sketch, which is also a table of $d \times w$ counters, where we set $d = \lceil \ln(\frac{1}{\delta}) \rceil$ and $w = \lceil \frac{\mathrm{e}}{\varepsilon} \rceil$ as well. The only difference is that CM-Sketch uses d (pairwise independent) random hash functions but LSH-Sketch uses d LSH functions. That explains why any optimization that can apply to CM-Sketch (e.g,. CU-Sketch [23] and CML-Sketch [51]) can also apply to LSH-Sketch. The key question then boils down to what LSH implementation we shall use. For reasons that we explain in section 3.2.4, we cannot use the classic implementation of (r_1, r_2, p_1, p_2)-sensitive family that we put forward in section 3.1.3. Instead, we adopt our own realization of (r_1, r_2, p_1, p_2)-sensitive family and the hash function $h(x)$ is:

$$h(x) = \left\lfloor \frac{a \cdot x + b}{r} \right\rfloor \tag{3.4}$$

where

1. a is a scalar from $a \sim U(1, \sqrt{2\varepsilon D/\mathrm{e}})$;
2. b is a scalar from $b \sim U(0, r)$, and
3. $r = \frac{D}{w}$.

Note that there are several differences with the classic implementation. First, the hash function now deals with scalar but not vector because data stream applications mainly focus on analyzing a single attribute at a time. Second, a is drawn from a *uniform distribution* instead of a *normal distribution*. Third, the quantization step r is related to the domain size D. We explain all these choices in detail after we discuss the error bound of LSH-Sketch in section 3.2.3.

3.2.1 Insertion

Inserting an item v from stream S into LHS-Sketch is the same as inserting an item into CM-Sketch. Algorithm 7 shows the pseudocode, where we use d preselected functions $L_1, L_2, \ldots, L_d$ that each, respectively, hashes a value to one of the w cells in the corresponding row. Each L_i function is defined on a function h_i randomly chosen from our (r_1, r_2, p_1, p_2)-*sensitive* family (Equation 3.4):

$$L_i(x) = h_i(x) \bmod w. \tag{3.5}$$

The $d \times w$ counters of LSH-sketch, $LSH[i][j]$, are all initialized with 0. For each item v, d counters (one for each row) get incremented by one (lines 2–4).

Algorithm 7 Insert-into-LSH-Sketch (v)

1: ▷ Each L_i is a preselected hash function that maps an item v to a cell in the ith row
2: **for** $i = 1$ to d **do**
3: $LSH[i][L_i(v)]$++;

The insertion complexity is the same as that of CM-Sketch, which is $O(1)$ because d and w are user-given space budget parameters.

3.2.2 Range-Counting: Algorithm AOC

Following the intuition we put forward in the beginning of this section, we have developed a range-counting algorithm, namely, Adjust Over-Counting (AOC), to support range-counting using LSH-Sketch.

Algorithm 8 shows the pseudocode of AOC. AOC carries out range-counting

row-by-row (lines 1–8). The crux of AOC is to ensure the counter of each cell is only considered once and we do so by adding a "touch-set" when processing each row (line 2). If there are multiple query values hashed into the same cell of the same row, the counter of that cell is only counted once towards the row counter (lines 4–8). The minimum of all d row counters is returned as the estimated count of the range query.

Algorithm 8 Adjust Over-Counting($Q[v_L, v_R]$)

1: **for** $i = 1$ to d **do**
2: Touch-Set $\mathbb{TS} = \varnothing$;
3: $row\text{-}count[i]$=0;
4: **for** each v in $[v_L, v_R]$ **do**
5: **if** $L_i(v)$ not in $\mathbb{TS}$ **then**
6: $row\text{-}count[i] += LSH[i][L_i(v)]$;
7: Insert $L_i(v)$ into $\mathbb{TS}$;
8: Return $\hat{f}_Q \leftarrow \min_i\{row\text{-}count[i]\}$;

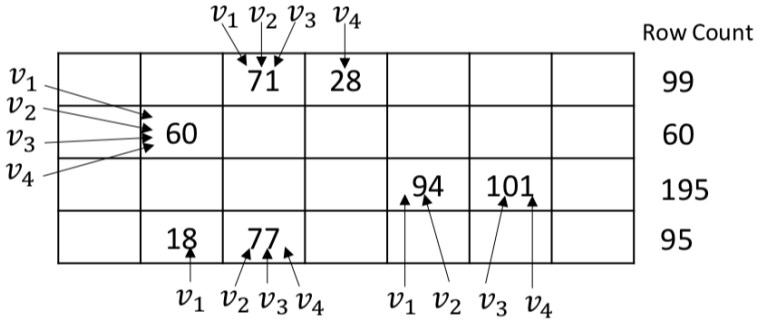


Figure 3.3: LSH-Sketch using AOC to do range counting

Figure 3.3 shows an example of processing a range-count query from values v_1 to v_4. For the first row, v_1, v_2, and v_3 all share the same bucket, and thus the row counter is $71 + 28 = 99$. The same is applied to each row, and finally the minimum among all $d = 4$ row counters, i.e., 60, is returned.

Lemma 14. *Given a range-count query $Q[v_L, v_R]$ whose range size is $|Q| = v_R - v_L + 1$, AOC returns answer with $O(|Q|\sqrt{w/D})$ latency.*

Proof. Based on (Equation 3.4), every $\frac{r}{a}$ consecutive values fall into the same hash bucket. So, for a range-count query Q, AOC needs to visit $\lceil \frac{|Q|}{r/a} \rceil + 1$ cells at most in each row. As we set $r = \frac{D}{w}$ and $a \sim U(1, \sqrt{2\varepsilon D/\mathrm{e}})$, so the number of cells AOC needs to visit in each row is at most

$$\left\lceil \frac{|Q|}{r/a} \right\rceil + 1 \leq \left\lceil \frac{\sqrt{2D/w}}{D/w} |Q| \right\rceil + 1.$$

As the number of rows, d, is a user-given space budget parameter, so d is a constant, and the query latency is $O(|Q|\sqrt{w/D})$. □

3.2.3 Accuracy

We aim to show that, using LSH-Sketch, there is a probability of at least $1 - \delta$ that the result $\hat{f}_Q$ of a range-count query is bounded by:

$$f_Q \leq \hat{f}_Q \leq f_Q + (4\varepsilon + (\sqrt{2\varepsilon \mathrm{e} D} + \mathrm{e})\frac{|Q|}{D})n.$$

We first study the error when there is only one row in LSH-Sketch (i.e., $d = 1$) and when the query is a point-count query (i.e., $|Q| = 1$) (Lemma 15). Then, we study the error when there is only one row in LSH-Sketch (i.e., $d = 1$) and when the query is a range-count query whose range has size $|Q| \geq 1$ (Lemma 16). Finally, we study the error when there are d rows in LSH-Sketch and when the query is a range-count query whose range has size $|Q| \geq 1$ (Theorem 2).

Lemma 15. *For LSH-Sketch with depth $d = 1$ and width w, a range-count query of size 1 (i.e., a point query on value v) using AOC has an approximate result $\hat{f}_v$, which is bounded by:*

$$f_v \leq \hat{f}_v \leq f_v + \frac{\lceil a \rceil}{a} \cdot \frac{n}{w}.$$

Proof. First, according to Equation 3.4, every r/a values are hashed to the same bucket. Given a domain D, there are then $\frac{D}{r/a}$ buckets. In LSH-sketch, there are only w cells (when $d = 1$), So, all those buckets are further hashed into w cells and thus each cell represents $\lceil \frac{D}{wr/a} \rceil$ buckets. As each bucket has r/a values at most, each cell represents $\lceil \frac{D}{wr/a} \rceil \frac{r}{a} = \lceil a \rceil \frac{r}{a}$ values at most. Given there are D values in the domain and $r = D/w$, the probability of collision is $\lceil a \rceil \frac{r}{a} / D = \frac{\lceil a \rceil}{aw}$. There are n values in the data stream, by linearity of expectation, the expected number of collisions is $\frac{\lceil a \rceil}{aw} n$. As every collision on the same cell increments the counter by 1, the expected error $\mathbf{E}(X)$ is thus also at most $\frac{\lceil a \rceil}{aw} n$.

□

Lemma 16. *For LSH-Sketch with $d = 1$ and width w, a range-count query of size $|Q|$ using AOC has an approximate result $\hat{f}_v$, which is bounded by:*

$$f_v \leq \hat{f}_v \leq f_v + (a+1)(\frac{2}{aw} + \frac{|Q|}{D})n.$$

Proof. Now we consider the error X on query Q when $d = 1$ for LSH-Sketch. Following Lemma 15, we know that, for each cell, the expected error is $\frac{\lceil a \rceil}{a} \frac{n}{w}$, which is at most $(1 + \frac{1}{a}) \frac{n}{w}$. In Lemma 14, we showed that the query visits $\lceil \frac{|Q|}{r/a} \rceil + 1$

cells at most. So, we have

$$\begin{aligned}
\mathbf{E}(X) &\leq \left(\left\lceil \frac{|Q|}{r/a} \right\rceil + 1\right)\left(1+\frac{1}{a}\right)\frac{n}{w} \\
&= \left(\left\lceil \frac{|Q|}{\frac{D}{aw}} \right\rceil + 1\right)\left(1+\frac{1}{a}\right)\frac{n}{w} \\
&\leq \left(\frac{|Q|}{\frac{D}{aw}} + 2\right)\left(1+\frac{1}{a}\right)\frac{n}{w} \\
&= \left(\frac{aw|Q|}{D} + 2\right)\left(\frac{a+1}{a}\right)\frac{n}{w} \\
&= (a+1)\left(\frac{2}{aw} + \frac{|Q|}{D}\right)n. \qquad (3.6)
\end{aligned}$$

□

Theorem 2. *For LSH-Sketch with $d = \lceil \ln(1/\delta) \rceil$ and $w = \lceil \mathrm{e}/\varepsilon \rceil$, a range-count query of size $|Q|$ using AOC has a probability of at least $1-\delta$ that the approximate result $\hat{f}_Q$ is bounded by:*

$$f_Q \leq \hat{f}_Q \leq f_Q + (4\varepsilon + (\sqrt{2\varepsilon \mathrm{e} D} + \mathrm{e})\frac{|Q|}{D})n.$$

Proof. When $w = \lceil \mathrm{e}/\varepsilon \rceil$, from Equation 3.6 we have

$$\mathbf{E}(X) \leq (a+1)(\frac{2\varepsilon}{\mathrm{e}a} + \frac{|Q|}{D})n.$$

By Markov's Inequality and $X = \hat{f}_Q - f_Q \geq 0$, we have

$$\mathbf{Pr}\left(X > (a+1)(\frac{2\varepsilon}{a} + \mathrm{e}\frac{|Q|}{D})n\right) \leq \frac{1}{\mathrm{e}}.$$

In other words, using each row i of LSH-Sketch to estimate the answer of Q, we have:

$$\mathbf{Pr}\left(\hat{f}_Q > f_Q + (a+1)(\frac{2\varepsilon}{a} + \mathrm{e}\frac{|Q|}{D})n\right) \leq \frac{1}{\mathrm{e}}.$$

As d locality sensitive hash functions are chosen independently for each row, then we have

$$\mathbf{Pr}\left(\hat{f}_Q \geq f_Q + (a+1)(\frac{2\varepsilon}{a} + \mathrm{e}\frac{|Q|}{D})n\right) \leq \frac{1}{\mathrm{e}^d}.$$

Since $\mathrm{e}^{-d} \leq \delta$, we have

$$\mathbf{Pr}\left(\hat{f}_Q \leq f_Q + (a+1)(\frac{2\varepsilon}{a} + \mathrm{e}\frac{|Q|}{D})n\right) \geq 1 - \delta. \tag{3.7}$$

As a is a scalar from $a \sim U(1, \sqrt{2\varepsilon D/\mathrm{e}})$, when $a = 1$, the error is $(4\varepsilon + 2\mathrm{e}\frac{|Q|}{D})n$. When $a = \sqrt{2\varepsilon D/\mathrm{e}}$, the error is $((1 + \frac{1}{\sqrt{2\varepsilon D/\mathrm{e}}})\varepsilon + (\sqrt{2\varepsilon \mathrm{e} D} + \mathrm{e})\frac{|Q|}{D})n$. Taking the worst case, the error is thus $(4\varepsilon + (\sqrt{2\varepsilon \mathrm{e} D} + \mathrm{e})\frac{|Q|}{D})n$ at most. □

3.2.4 Implementation of (r_1, r_2, p_1, p_2)-sensitive family for LSH-Sketch

Now we discuss the design choice of our implementation of (r_1, r_2, p_1, p_2)-sensitive family. First, following Equation 3.7, we estimate the expected error $\mathbf{E}(X)$ by[2]

$$\mathbf{E}(X) \approx (a+1)\left(\frac{2\varepsilon}{a} + \mathrm{e}\frac{|Q|}{D}\right)n. \tag{3.8}$$

Now consider different range size $|Q|$:

[2]Following equation 3.6, we can also get the lower bound that $\mathbf{E}(X) \geq (a+1)(\frac{1}{aw} + \frac{|Q|}{D})$, which is close to the upper bound in Equation 3.7. So here we use the result of Equation 3.7 to estimate it.

`Pitfall` 1: When $|Q| \ll D$, $\frac{|Q|}{D} \approx 0$, then we have

$$\mathbf{E}(X) \approx (a+1)(\frac{2\varepsilon n}{a}).$$

So, when $a \to 0$, $\mathbf{E}(X) \to \infty$.

`Pitfall` 2: When $|Q| \approx D$, $\frac{|Q|}{D} \approx 1$, then we have

$$\mathbf{E}(X) \approx (a+1)\left(\frac{2\varepsilon}{a} + \mathrm{e}\right) n.$$

Thus, either $a \to \infty$ or $a \to 0$ will lead to $\mathbf{E}(X) \to \infty$.

The two pitfalls above explain why we cannot adopt the original implementation of (r_1, r_2, p_1, p_2)-sensitive family that sets $a \sim N(0,1)$ because that gives the possibilities of $a \to 0$ or $a \to \infty$. Notice that the use of normal distribution is necessary when LSH is used in **dimension reduction** because normal distribution is a p-stable distribution that can preserve the locality after projecting onto the lower dimensional space [21]. In our streaming applications, the requirement of drawing from a p-stable distribution, however, is unnecessary because stream data is just 1-dimensional and any linear operation on them can preserve the locality. In other words, we can also pick non p-stable distributions including the uniform distribution.

Now let us rearrange Equation 3.8 in order to make informed decision for the range of a from uniform distribution:

$$\mathbf{E}(X) \approx 2\varepsilon n + 2\varepsilon n/a + (a+1)\mathrm{e}|Q|n/D.$$

The first term in the above sum is a constant with respect to a. The second term increases when a decreases. The third term increases when a increases. Therefore, to minimize $\mathbf{E}(X)$, we shall balance the effect of the second and the third terms, i.e.,

$$2\varepsilon/a = (a+1)\mathrm{e}|Q|/D,$$

If $a \geq 1$, then we know $a^2 < a(a+1)$, so we can have

$$\begin{aligned} 2\varepsilon D &= a(a+1)\mathrm{e}|Q| \\ a(a+1) &= 2\varepsilon D/(\mathrm{e}|Q|) \\ a^2 &\leq 2\varepsilon D/(\mathrm{e}|Q|) \end{aligned}$$

$|Q|$ is workload dependent. If the workload characteristic is unknown, we consider the worst case where $|Q| = 1$. So, we have an upper bound of a as

$$a \leq \sqrt{2\varepsilon D/\mathrm{e}}.$$

From `Pitfall` 1 and `Pitfall` 2, a could be anything but 0 or ∞. So, overall we set $a \sim U(1, \sqrt{2\varepsilon D/\mathrm{e}})$.

Next we show that the family of hash functions used in our own LSH implementation (Equation 3.4) is (r_1, r_2, p_1, p_2)-sensitive. Especially,

Theorem 3. *The family of functions h, in the form of $h(x) = \lfloor \frac{ax+b}{r} \rfloor$, with $a \sim U(1, \sqrt{2\varepsilon D/\mathrm{e}})$, $b \sim U(0, r)$, and $r = D/w$ is $(\frac{\sqrt{2}r}{4}, 0.8r, 0.5, 0.2)$-sensitive.*

Proof. First, for two random variables x, y drawn from the 1D domain, let $dist(x, y)$ denote the distance between them, we have:

$$\mathbf{Pr}(\lfloor x \rfloor = \lfloor y \rfloor) = \begin{cases} 1 - dist(x, y) & \text{if } 0 \leq dist(x, y) < 1 \\ 0 & \text{if } dist(x, y) \geq 1 \end{cases} \tag{3.9}$$

Next, for two values v_1, v_2, we have

$$dist(\frac{av_1 + b}{r}, \frac{av_2 + b}{r}) = \frac{a}{r} dist(v_1, v_2).$$

So, we have the collision probability of our LSH implementation as:

$$\mathbf{Pr}(h(v_1) = h(v_2)) = \mathbf{Pr}(\lfloor \frac{av_1 + b}{r} \rfloor = \lfloor \frac{av_2 + b}{r} \rfloor)$$
$$= \begin{cases} 1 - \frac{a}{r} dist(v_1, v_2) & \text{if } 0 \leq \frac{a}{r} dist(v_1, v_2) < 1 \\ 0 & \text{if } \frac{a}{r} dist(v_1, v_2) \geq 1 \end{cases}$$

We consider two cases:

- When $dist(v_1, v_2) \leq \frac{\sqrt{2r}}{4}$, we have

$$\mathbf{Pr}(h(v_1) = h(v_2)) \geq 1 - \frac{a}{r} dist(v_1, v_2)$$
$$\geq 1 - \frac{a}{r} \frac{\sqrt{2r}}{4}$$

As $a \sim U(1, \sqrt{2\varepsilon D/\mathrm{e}})$, we have

$$a \leq \sqrt{2\varepsilon D/\mathrm{e}} \approx \sqrt{2D/w} = \sqrt{2r}. \tag{3.10}$$

So we have

$$\begin{aligned}\mathbf{Pr}(h(v_1) = h(v_2)) &\geq 1 - \frac{\sqrt{2r}}{r}\frac{\sqrt{2r}}{4} \\ &\geq 1 - 0.5 = 0.5.\end{aligned}$$

- When $dist(v_1, v_2) \geq 0.8r$, we have

$$\begin{aligned}\mathbf{Pr}(h(v_1) = h(v_2)) &\leq \max\{1 - \frac{a}{r}dist(v_1, v_2), 0\} \\ &\leq \max\{1 - \frac{a}{r}0.8r, 0\}.\end{aligned}$$

As $a \sim U(1, \sqrt{2\varepsilon D/\mathrm{e}})$, we have $a \geq 1$, then

$$\begin{aligned}\mathbf{Pr}(h(v_1) = h(v_2)) &\leq \max\{1 - \frac{1}{r}0.8r, 0\} \\ &\leq \max\{1 - 0.8, 0\} = 0.2.\end{aligned}$$

□

3.3 Experiment

We evaluate the use of AOC over LSH-Sketch to answer range-count queries and compare it against the two baseline approaches we put forward in section 3.1.2. Specifically, we used CM-Sketch to carry out range-counting (section 3.1.2.1) and we also built multi-level of CM-Sketches (section 3.1.2.2). We evaluate the accuracy (section 3.2.3), insertion throughput (section 3.2.1), and query throughput (section 3.2.2) of three approaches over three real data sets and three

synthetic data sets under different data skewness.

- **Kosarak Click Stream** — Anonymized click-stream from Hungarian online news portal[3]. This data set has been used in [10, 44, 52] and contains $8M$ records and the domain size $D = 41270$.

- **WebDocs** — Transactional data set from the web collection[3]. It has been used in [10] and contains $15M$ records and the domain size $D = 580947$.

- **CAIDA Trace 2018** — Anonymized passive traffic traces[4]. It has been used in [31] and contains $30M$ flow records and the domain size $D = 10^{13}$.

- **Zip-0, Zip-1, Zip-2** — Synthetic data sets with stream size $32M$ and the domain size $D = 8M$ under three different skew factors 0 (uniform), 1 (mid-skew), 2 (high-skew). The stream size and domain size settings were used in [52] as well.

The experiment platform is a 3.2 GHz Intel i5 CPU with 8GB RAM. All the methods are implemented in C++ and compiled with -O3 optimization. Following [66], we set the default space budget as 1MB. The default value for d is 4, yielding a 98% probability guarantee on the error bound [11]. Under the same space budget and the same value of d, the value of w is set accordingly. Each experiment is repeated 100 times and we report the averages. For ease of discussion, we label using AOC over LSH-Sketch simply as LSH-Sketch, the approach of range-counting over CM-Sketch (section 3.1.2.1) as CM-Sketch, and

[3] http://fimi.ua.ac.be/data/
[4] http://www.caida.org/data/overview

the approach of range-counting over multi-level of CM-Sketches (section 3.1.2.2) as CMS-ML.

3.3.1 Accuracy

In this experiment, we evaluate the accuracy of all methods under different query range sizes. We generate queries of range from D (the whole domain), $D/2$, $D/4$, ..., 2, 1 (point-count query). We report both *absolute error* and *relative error* of the query results.

Figure 3.4 to 3.6a show the absolute errors of all methods on all data sets (in log-log scale). Empirically, the absolute errors of CM-Sketch and CMS-ML increase when the query range size increases, which are consistent with our theoretical results listed in Table 1.1. The empirical errors of LSH-Sketch are better than the theoretical errors. Specifically, the absolute errors of LSH-Sketch do not grow with the query range size on all real data sets. When $|Q| = D$, it is essentially asking the total number of values that have been inserted into the sketch so far. As algorithm AOC counts each cell only once, AOC could return the exact answer in this case and has 0 error. When $|Q| = 1$, i.e., point-count query, LSH-Sketch also outperforms CM-Sketch (and CMS-ML) on all data sets except Kosorak Click Stream where CM-Sketch is marginally better than LSH-Sketch in that data set.

For range counting, the relative error of a query is even more important because the count of a range query naturally increases with the query range size. Figure 3.4b, Figure 3.5b and Figure 3.6b thus show the relative errors of all methods on all data sets (in log-log scale). Empirically, we see that LSH-Sketch

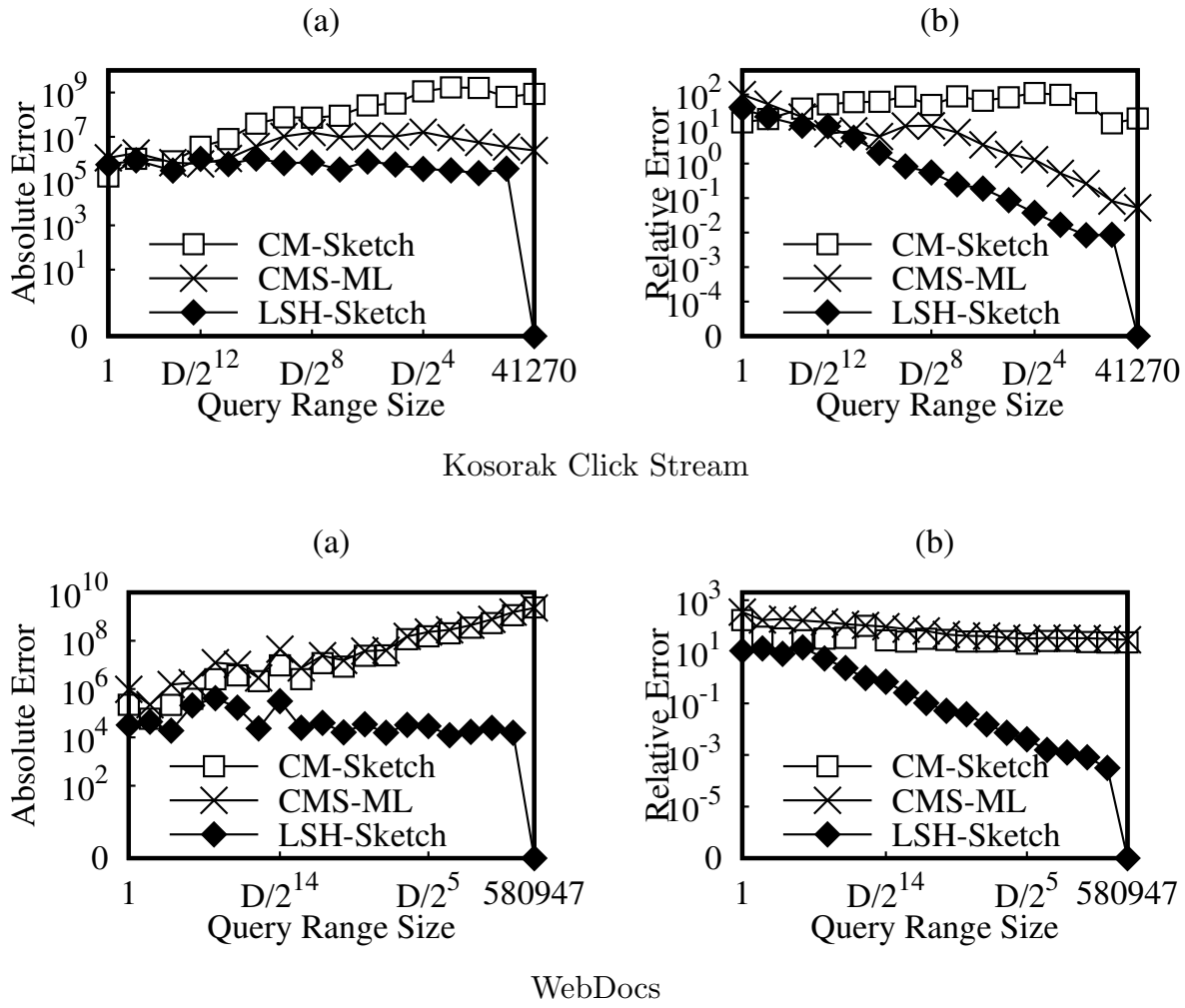


Figure 3.4: Range Count Accuracy (log-log scale) on Kosorak and WebDocs datasets

generally outperforms CM-Sketch and CMS-ML in terms of relative errors on all data sets and all query ranges. Furthermore, although the theoretical error bound of CMS-ML is better than CM-Sketch, we observe that empirically CMS-ML is not always better than CM-Sketch on range-counting.

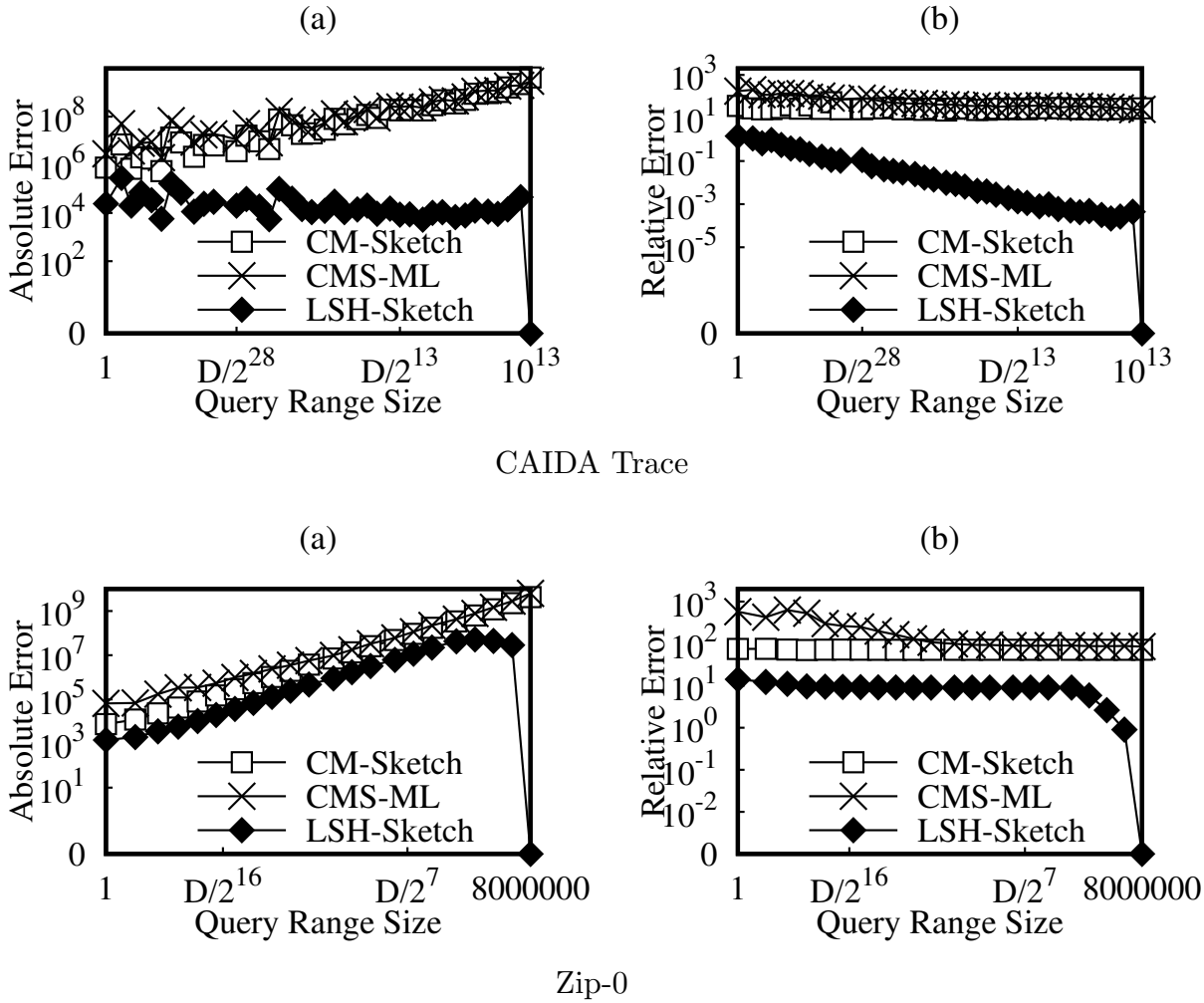


Figure 3.5: Range Count Accuracy (log-log scale) on CAIDA and Zip-0 datasets

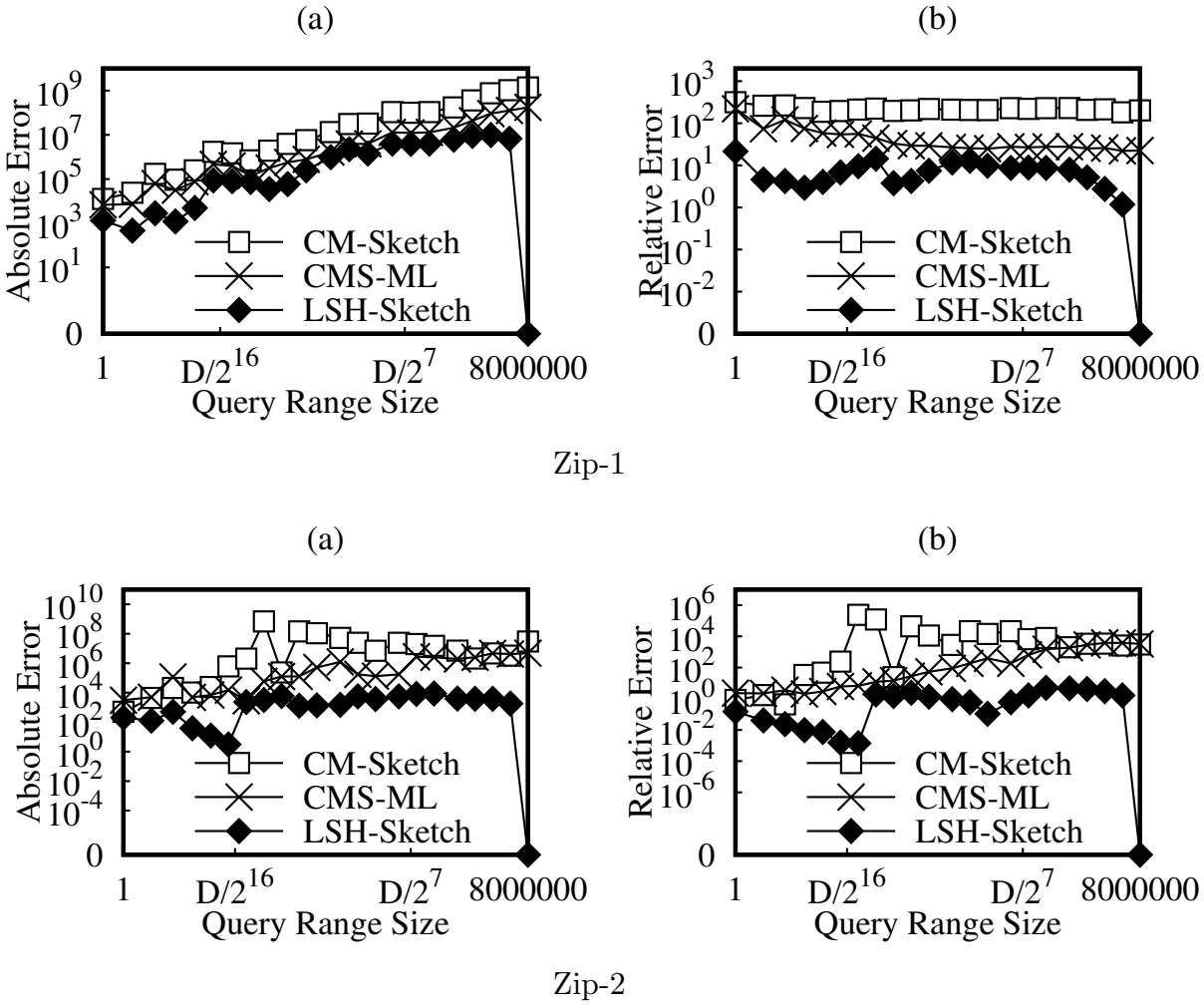


Figure 3.6: Range Count Accuracy (log-log scale) on Zip-1 and Zip-2 datasets

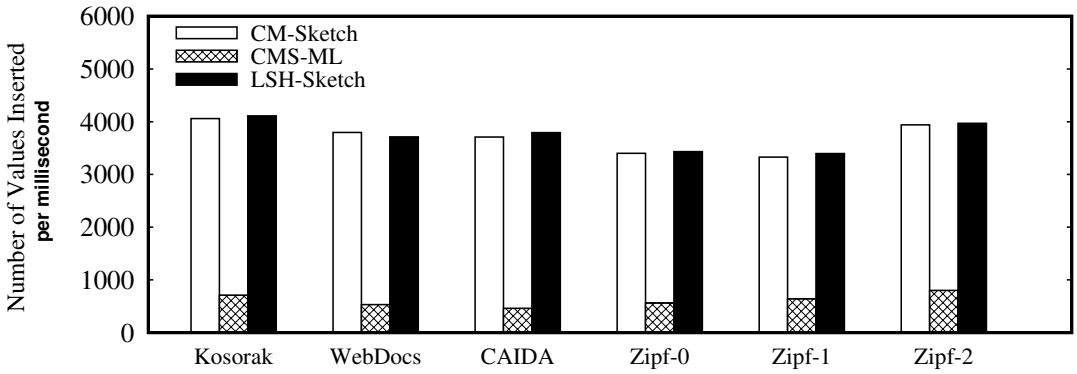


Figure 3.7: Insertion Throughput

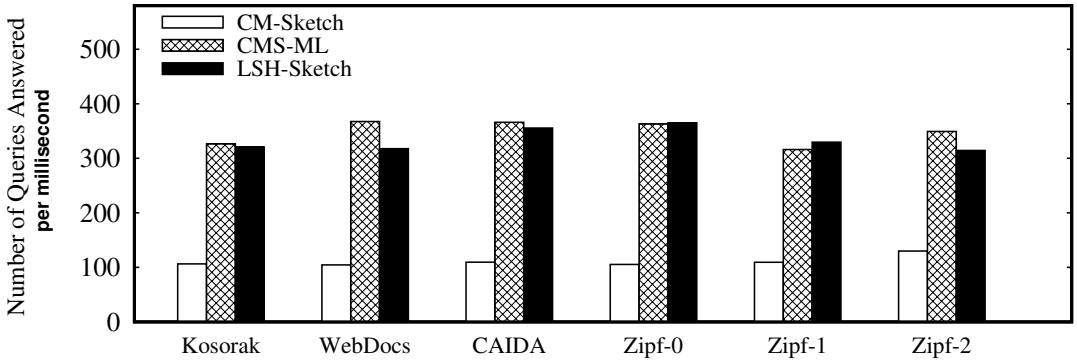


Figure 3.8: Query Throughput

3.3.2 Insertion Throughput

In this experiment, we evaluate the insertion throughput of all methods on all data sets. Following [66], we report the insertion throughput as the number of values could be inserted per millisecond.

Figure 3.7 shows the empirical results. Consistent with the theoretical results listed in Table 1.1, CM-Sketch and LSH-Sketch have much better insertion throughput than CMS-ML because they do not need to maintain multiple

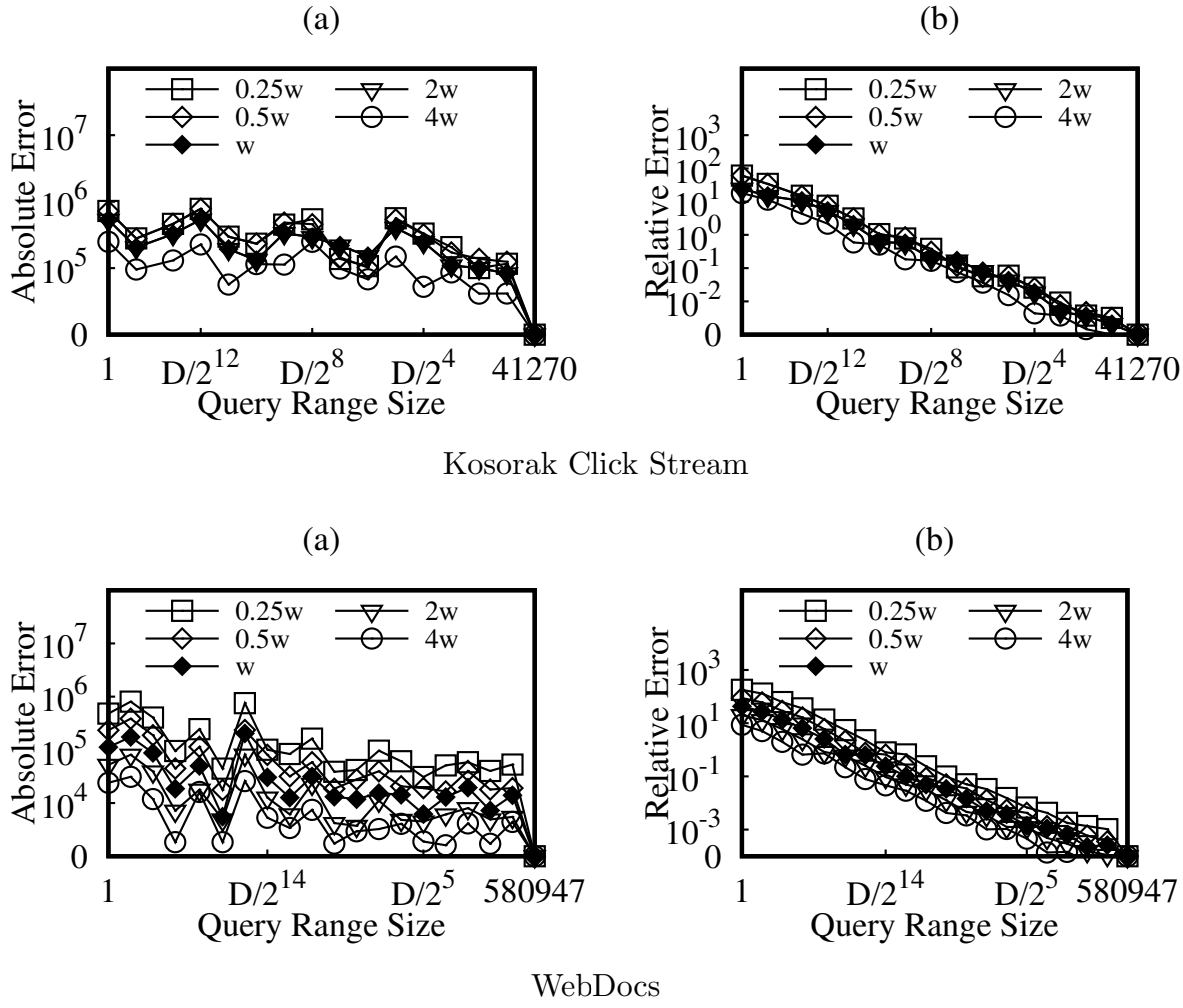


Figure 3.9: Vary the Width of LSH-Sketch (log-log scale) on Kosorak and WebDocs Datasets

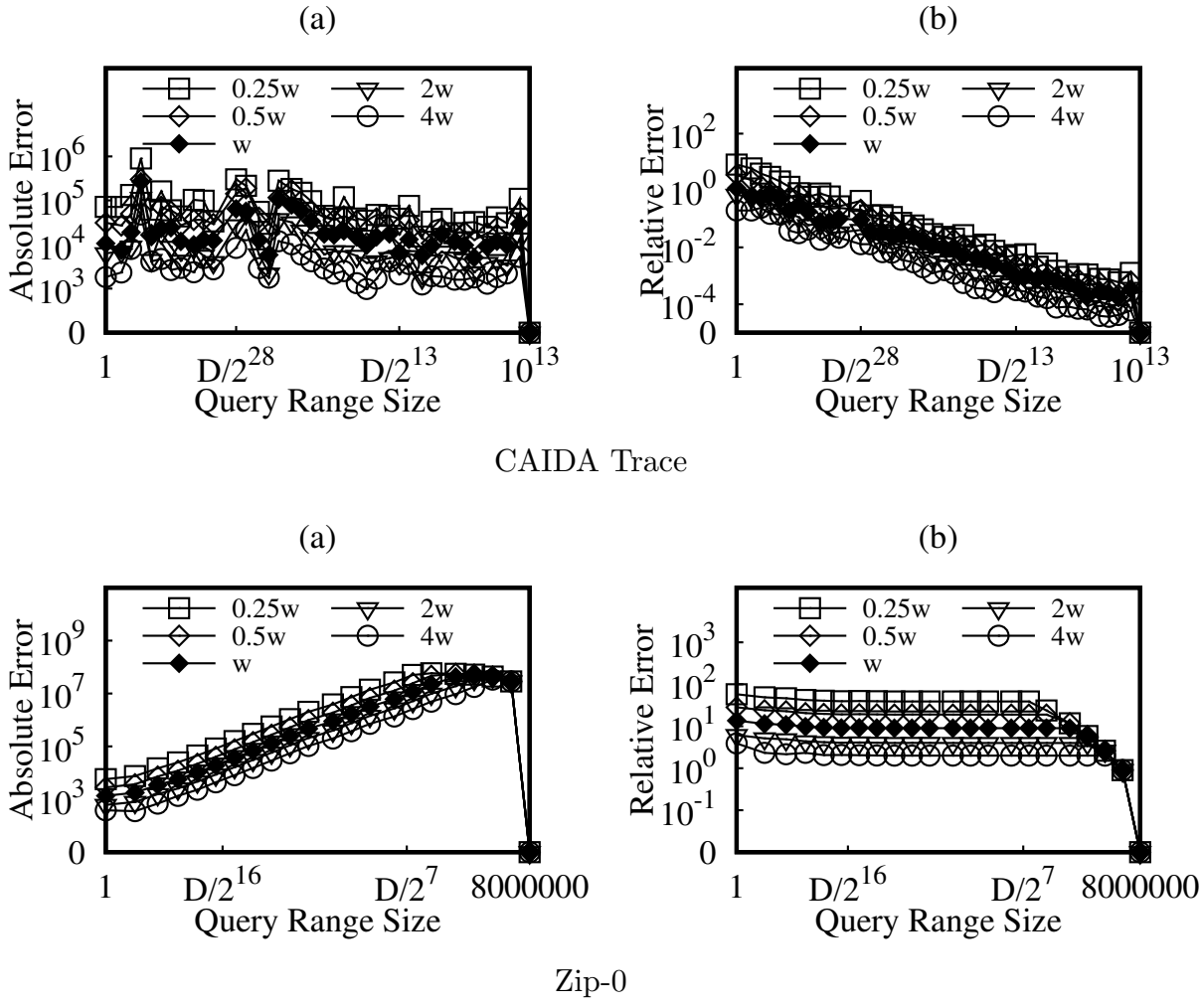


Figure 3.10: Vary the Width of LSH-Sketch (log-log scale) on CAIDA and Zip-0 Datasets

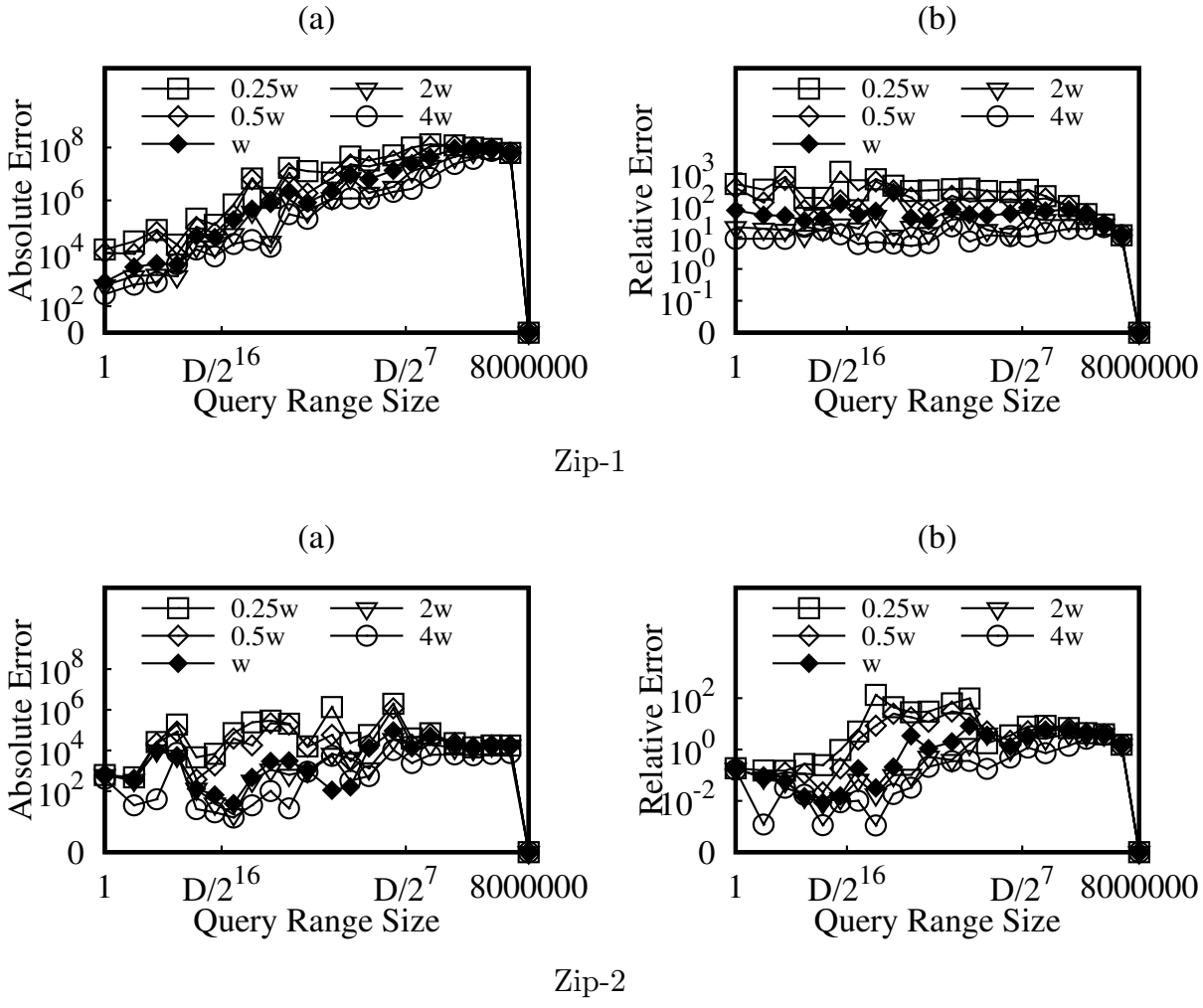


Figure 3.11: Vary the Width of LSH-Sketch (log-log scale) on Zip-1 and Zip-2 Datasets

sketches. Empirically, both the insertion throughput of CM-Sketch and LSH-Sketch are about 4.9× to 8.2× of CMS-ML. Since both CM-Sketch and LSH-Sketch have $O(1)$ insertion latency, their empirical insertion throughputs are also similar.

3.3.3 Query Throughput

In this experiment, we evaluate the query throughput of all methods on all data sets. We generated a workload of queries in random range size and then used the same workload on all three methods and reported how many queries each method can answer per millisecond.

Figure 3.8 shows the empirical results. Consistent with the theoretical results listed in Table 1.1, CMS-ML and LSH-Sketch have much better query throughput than CM-Sketch because CM-Sketch query latency is $O(|Q|)$ whereas CMS-ML and LSH-Sketch are $O(\log |Q|)$ and $O(|Q|\sqrt{w/D})$, respectively. Empirically, both the query throughput of CMS-ML and LSH-Sketch are about 2.4× to 3.5× of CM-Sketch. CMS-ML has slightly better query throughput than LSH-Sketch except on Zip-0 and Zip-1 data sets. Nonetheless, CMS-ML indeed has unacceptable insertion throughput and much poorer accuracy than LSH-Sketch.

3.3.4 Accuracy and Space Trade-off

Finally, we carried out two more experiments to study the query accuracy and space trade-off of LSH-Sketch. Specifically, we varied the space budget by changing the values of sketch width w and sketch depth d separately.

Figure 3.9 to Figure 3.11 show the query accuracy when we vary the sketch width from $0.25w$ to $4w$. In this experiment, the sketch depth d remains as the default value 4. So when we increase the width to $4w$, the space budget is increased $4\times$. Similarly, when we decrease the width to $0.25w$, the space budget is decreased to $\frac{1}{4}$ of the default space budget. From the result, we observe an improvement on the accuracy (error) when more space is given and the opposite happens when less space is given, which is usual and under our expectation. This result is also generalizable to all query ranges.

Figure 3.12 to Figure 3.14 show the query accuracy when we vary the sketch depth from $0.25d$ to $4d$ where $d = 4$. In this experiment, the sketch width w remains the same and set as the value when assuming $d = 4$. So, when we increase the depth to $4d$, the space budget is increased $4\times$. Similarly, when we decrease the depth to $0.25d$ (i.e. $d = 1$), the space budget is decreased to $\frac{1}{4}$ of the default space budget. From the result, we observe an insignificant difference on the errors among different d values. That is because when $d = 1$, we have already obtained a probability of 0.63 that the error bound holds. Empirical results show that it might be good enough already when comparing with $d = 16$ (i.e., $4d$), whose probability is 0.998 that the error bound holds.

3.4 Related Work

Sketches have been increasingly used in many applications to provide statistics under provable resource-accuracy trade-off. Recent studies [24, 25, 39, 40, 62, 71] show that sketches need to support queries beyond simple point-count and heavy-hitters [42] and range-count query is one of them [31].

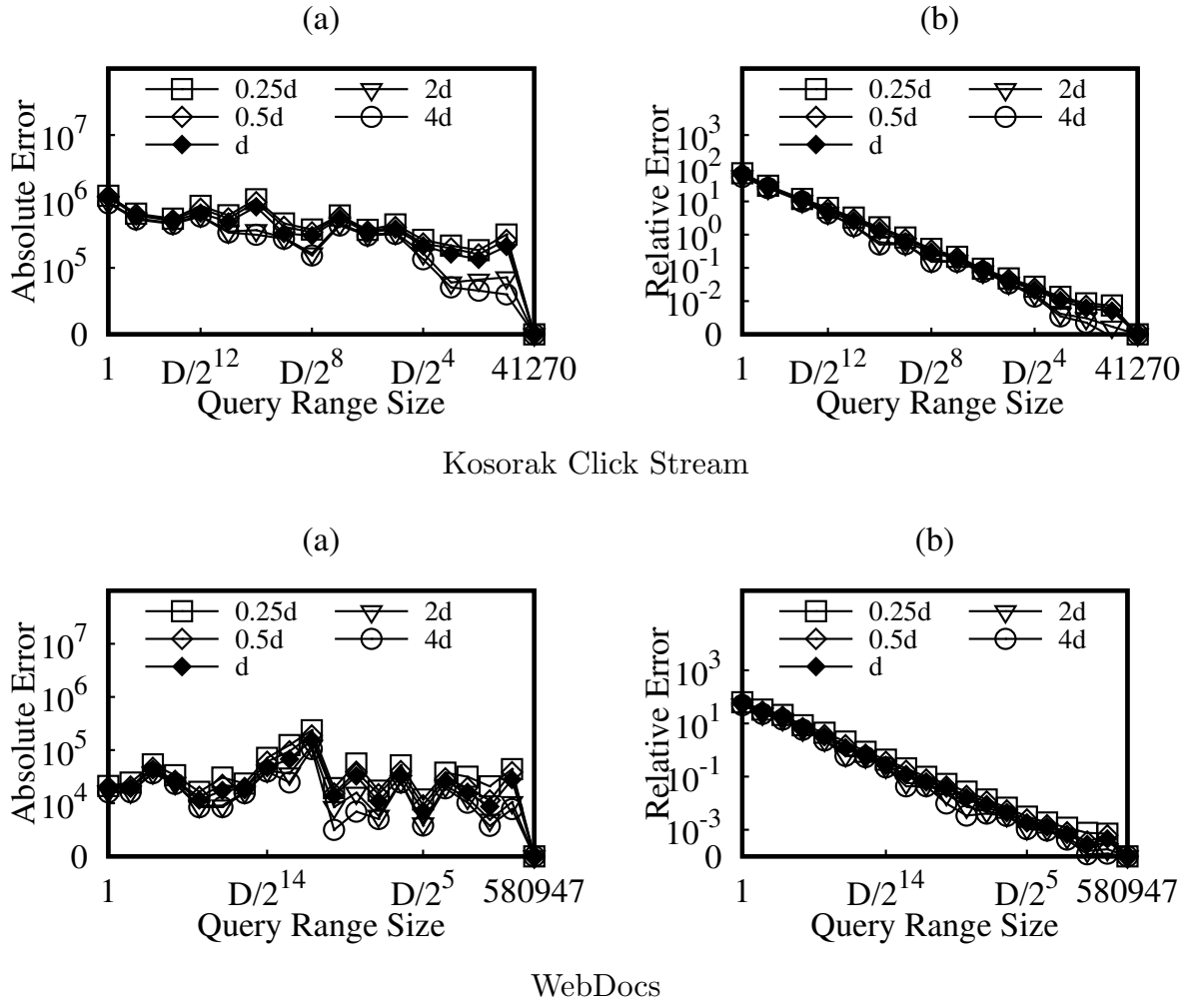


Figure 3.12: Vary the Depth of LSH-Sketch (log-log scale) on Kosorak and WebDocs Datasets

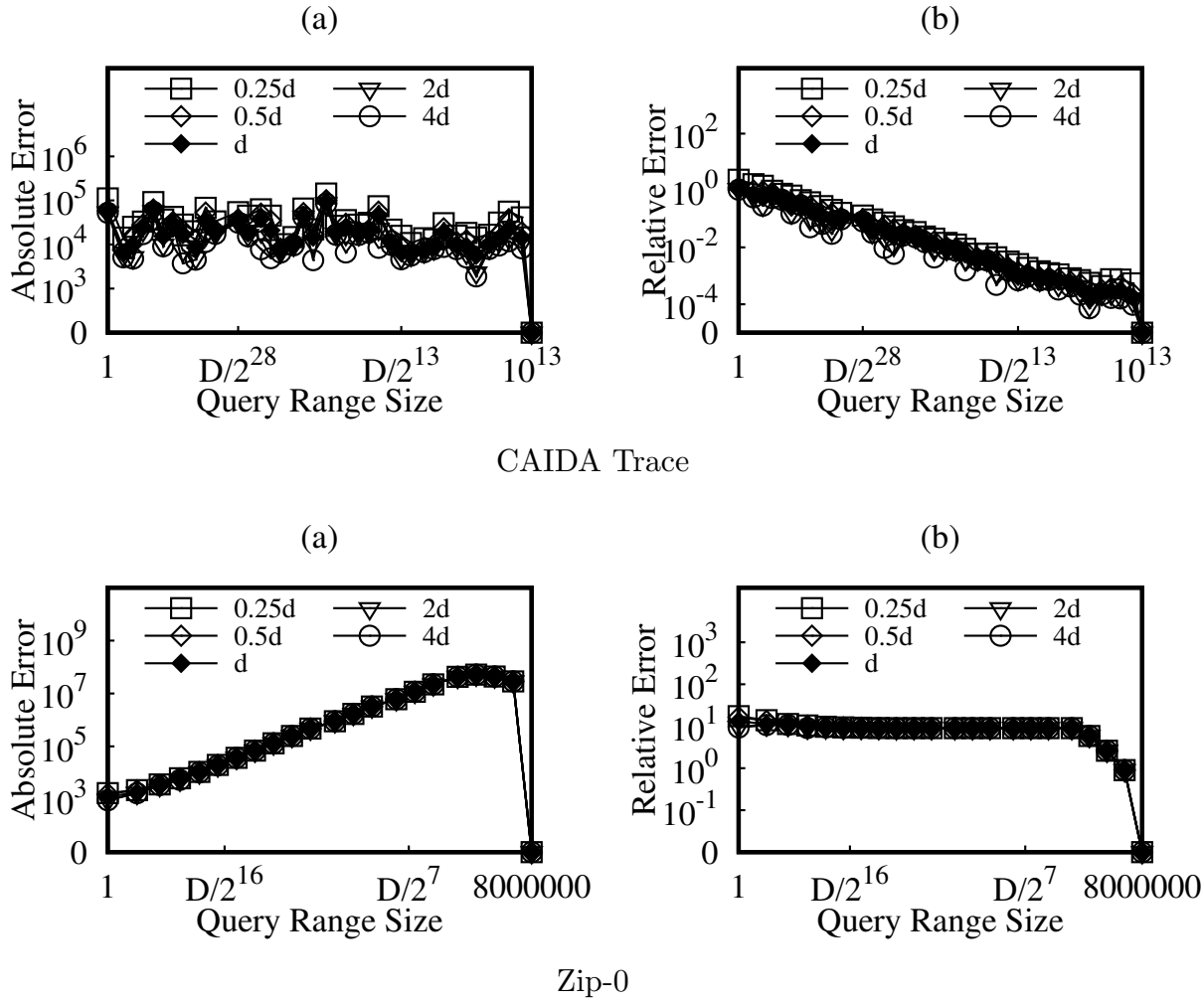


Figure 3.13: Vary the Depth of LSH-Sketch (log-log scale) on CAIDA and Zip-0 Datasets

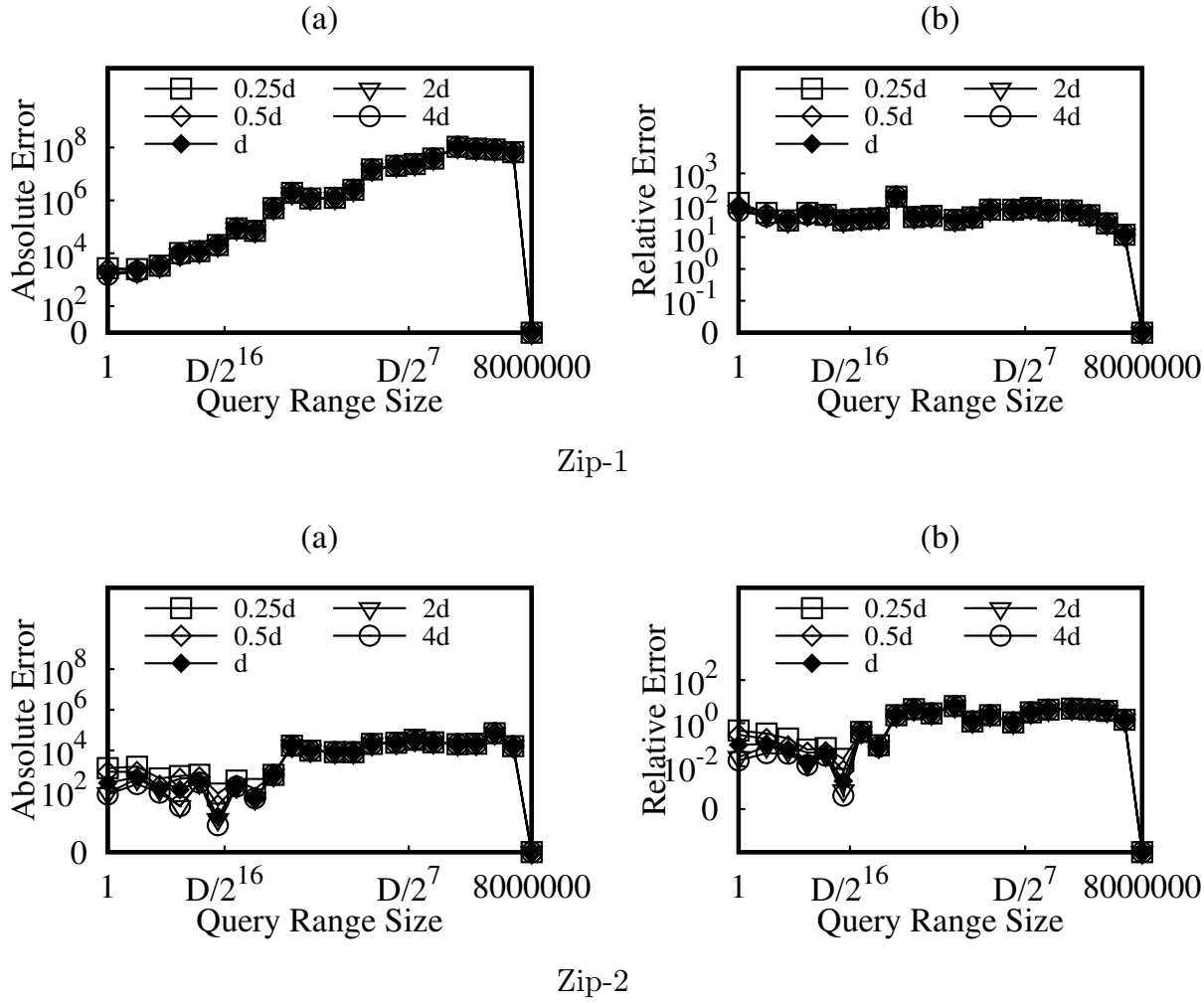


Figure 3.14: Vary the Depth of LSH-Sketch (log-log scale) on Zip-1 and Zip-2 Datasets

Many sketches today are specifically designed for point-count queries and they are based on or inspired by CM-Sketch [52, 65, 66]. For example, CU-Sketch [23] and CML-Sketch [51] aim to improve the accuracy by updating only the row with minimum count (instead of all d rows) and updating only some rows with certain probabilities, respectively. To overcome the error induced by skewed data, Pyramid Sketch [66] allocates more space for counters of hot items and less space for counters of cold items. Augmented Sketch [52] mitigates error from data skewness by pre-filtering the heavy hitters into a list and inserting the rest into a CM-Sketch. In terms of implementation, Augmented Sketch is optimized for modern CPU architectures (e.g., use of SIMD instructions) so that it has good query and insertion throughput. Elastic Sketch [65] further proposes techniques to compress CM-Sketch so that the sketches can still be transferred between the data plane and control plane when the network bandwidth is largely reduced due to congestion or attack. Since LSH-Sketch shares the same structure with CM-Sketch, all the accuracy and performance optimization techniques above should be applicable to LSH-Sketch and we regard those as our important future work.

There are sketches specifically designed for heavy-hitter queries (e.g., space saving [46], USS [60]). Since they offer no error bound for values other than the heavy-hitters, they are not suitable for range-count queries. Building multi-level of sketches for range-count was first discussed in the original CM-Sketch paper [16]. A recent work that requires range-counting has adopted that multi-level sketches approach [31]. Their focus is on how to achieve better accuracy by automatically tuning the sizes of different levels (via the sketch width w and sketch depth d) under the same space budget. LSH-Sketch can directly solve their range-counting problem as our empirical results show that LSH-Sketch is

way better than the multi-level approach in terms of accuracy and insertion throughput.

Concerning range-counting, histograms like V-Optimal [29,32] and DigitHist [55] can provide good accuracy but they require multiple passes on the data in order to gain error bound [12]. SuRF [70] can support approximate range-counting but it was designed for static data and insertions require a full rebuild of the whole data structure, which is not suitable for data stream.

3.5 Summary

In this chapter, we study range-counting problem over large sequence data of large velocity, i.e., rapid data stream. A new sketch based on Locality Sensitive Hashing, LSH-Sketch, is proposed and we show that it has good accuracy and throughput, from the perspective of both theorems and empirical evaluations.

Our LSH-Sketch serves as a core and there are various optimisations on it can be done in the future work. For example, we may study whether the advanced techniques based on CM-Sketch, e.g., Pyramid Sketch framework [66], can be applied to our LSH-Sketch directly. If not, investigate the bottleneck and work on how other kinds of optimisation methods can be built on.

Chapter 4

Conclusion and Future Work

In this thesis, for large sequence data of huge volume, we introduce the notion of historic moment, which complements existing work [33, 69] and provides holistic insights from sequence data. The computational issue of historic moment focuses on the incremental and interactive aspects, in which new data are expected to arrive regularly and users are supposed to discover new historic moments right away, by feeding in different input parameters. To this end, we present SOIA, a highly-efficient incremental algorithm using minimal space. Space-optimality is important for online analysis and real-time monitoring systems because that significantly reduces the index size, thereby reducing the I/O per operation or making the index memory-resident even when there are many data sequences. Case studies show that historic moments are helpful in computational journalism as well as in seismology. Experimental studies show that SOIA is both space and time efficient and outperforms the baseline on real data. In future work, we are going to study the multi-dimension case of historic mo-

ments, which is a more general concept compared with this thesis. More kinds of definitions on historic moments, e.g., define it according to "similarity" only, are to study and to evaluate whether they could give more insightful stories.

For range-count queries posed on rapid data stream, we present LSH-Sketch. LSH-Sketch is more general than point-count sketches because it supports both range-counting and point-counting. LSH-Sketch outperforms CM-Sketch in terms of accuracy and query throughput both theoretically and empirically. LSH-Sketch's insertion throughput is on a par with CM-Sketch. As CM-Sketch is the core of many sketch variants and applications on top, LSH-Sketch has the potential to replace CM-Sketch in multiple domains [7, 19, 23, 26, 38, 51–53, 65, 66]. We plan to explore all those as our future work, from both theoretical and empirical evaluations.

Appendices

Appendix A

Appendix

The main discussion focuses on finding historic moments in one data sequence. In this appendix, we also extend our problem and the algorithms to multiple data sequences (Note that in [69], the authors also study how to find prominent streaks in multiple data sequences). We first give a motivating example from a piece of sports news in April 2014[1]:

> *"He (Kevin Durant) put up 25 points in his 40th consecutive game, which is the longest streak since Michael Jordan scored 25 in 40 consecutive games in the 1986-87 season. Wilt Chamberlain, who scored 25 or more in 80 consecutive games in 1960-61, holds the all time record in that category. "*

In this piece of news, for what streak Kevin Durant has achieved, the reporter reports historic moments from Michael Jordan and Wilt Chamberlain,

[1] http://ftw.usatoday.com/2014/04/kevin-durant-michael-jordan-record-thunder

which are two other data sequences.

Definition 11. Top-k Situational Streak in Multiple Sequences. *Given multiple sequences $\mathbb{D}_n = \{D_n^1, \cdots, D_n^w\}$ that contain w data sequences and each has n values, let $\mathcal{SS}_n^{D_n^1}, \cdots, \mathcal{SS}_n^{D_n^w}$ be the set of situational streaks for each data sequence respectively. Let $\mathbb{SS}_n = \mathcal{SS}_n^{D_n^1} \cup \cdots \cup \mathcal{SS}_n^{D_n^w}$ be all situational streaks in $\mathbb{D}_n$. The top-k situational streaks in $\mathbb{D}_n$ are the k streaks in $\mathbb{SS}_n$ with the highest values.*

Definition 12. Analogous Streak in Multiple Sequences. *Given multiple sequences $\mathbb{D}_n = \{D_n^1, \cdots, D_n^w\}$, let $\mathcal{LPS}_n^{D_n^1}, \cdots, \mathcal{LPS}_n^{D_n^w}$ be the local prominent streaks for each sequence respectively. Let $\mathbb{LPS}_n = \mathcal{LPS}_n^{D_n^1} \cup \cdots \cup \mathcal{LPS}_n^{D_n^w}$ be all local prominent streaks set in $\mathbb{D}_n$. A local prominent streak $s \in \mathbb{LPS}_n$ is an* analogous streak *of a situational streak $z \in \mathbb{SS}_n$ when:*

1. *$s.j < z.i$ (i.e., s ends before z starts),*
2. *$|s| \geq |z| \cdot \sigma$ (i.e., the length of s is at least σ times that of z, where $\sigma \geq 0$ is a similarity threshold), and*
3. *$s.v \geq z.v \cdot \sigma$ (i.e., the value of s is at least σ times that of z).*

Definition 13. Historic Moments in Multiple Sequences. *Given multiple sequences $\mathbb{D}_n = \{D_n^1, \cdots, D_n^w\}$, a similarity threshold σ and a positive integer k. For each of situational streak z in top-k $\mathbb{SS}_n$, let $\mathbb{AS}(z)$ be the set of analogous streaks in multi-sequence $\mathbb{D}_n$ for z. Assume that each streak s in $\mathbb{AS}(z)$ is represented by a 4D point $(|s|, s.j, s.v, x)$, where x indicates which data sequence D^x that s comes from. The* historic moments *with respect to z, denoted by $\mathbb{HM}(z)$, is the skyline of $\mathbb{AS}(z)$ with respect to the* **first three dimensions**, *i.e., $|s|$, $s.j$, and $s.v$ of streak s.*

In the sports news above, each data sequence in $\mathbb{D}_n$ represents the points that the player gets. On 5 April 2014, the streak with value 25 and length 40 from Kevin Durant is the top-1 situational streak. Its historic moments include the streak with value 25 and length 40 from Michael Jordan in 1986, as well as the streak with value 25 and length 80 from Wilt Chamberlain in 1960.

BIA and SOIA can be adapted to deal with this multi-sequence situation in a straightforward manner. We name them as BIA-MS and SOIA-MS, respectively.

A.0.1 BIA-MS

BIA-MS is largely similar to BIA that $\mathbb{LPS}_n - \mathbb{SS}_n$ are inserted into the same R-tree, each $\mathcal{SS}_n^{D_n^i}$ is stored separately, and we have:

1. Maintenance. Each sequence of new values $\langle v_{n+1}^{D^i}, v_{n+2}^{D^i} \cdots v_m^{D^i} \rangle$ for D^i calls the the maintenance procedure of BIA, i.e., Algorithm 2, once.

2. Look up. Same as the look up procedure of BIA, i.e., Algorithm 3.

A.0.2 SOIA-MS

SOIA-MS is largely similar to SOIA. SOIA-MS is also space-optimal. as it keeps minimal subset $\mathbb{U}_n$ of $\mathbb{LPS}_n$. It mainly generalizes the notion of *perplexing streaks* and *non-perplexing streaks* for multiple sequences:

Definition 14. Perplexing Streak. *A streak $p \in \mathbb{LPS}_n$ is a* perplexing streak *when there exists a situational streak $z \in \mathbb{SS}_n$ such that :*

1. *p overlaps with z,*

2. $|p| \geq |z| \cdot \sigma$, *and*

3. $p.v \geq z.v \cdot \sigma$.

We use $\mathbb{P}_n$ *to denote the set of all perplexing streaks in* $\mathbb{D}_n$.

Definition 15. NON-PERPLEXING STREAK. *Streaks in* $\mathbb{LPS}_n$ *that are not in* $\mathbb{P}_n$ *are* non-perplexing *streaks, denoted as* $\mathbb{N}_n$. *That is,* $\mathbb{N}_n = \mathbb{LPS}_n - \mathbb{P}_n$. *A streak* $s \in \mathbb{LPS}_n$ *is a non-perplexing streak if, for every situational streak* $z \in \mathbb{SS}_n$ *that* s *overlaps with, either* $|s| < |z| \cdot \sigma$, *or* $s.v < z.v \cdot \sigma$

Theorem 4. *The minimal subset* $\mathbb{U}_n$ *of* $\mathbb{LPS}_n$ *is the set of streaks in* $skyline(\mathbb{N}_n) \cup \mathbb{P}_n$ *that (i) are not universally dominated by any streak in* $\mathbb{P}_n$ *and (ii) have length at least* σ.

Proof. We prove the theorem by adapting the corresponding proof in Theorem 1. This is done by proving Lemma 17 (the if case) and Lemma 18 (the only if case) below.

Lemma 17. *If a streak* s *can serve as a historic moment of some streak in* $\mathbb{SS}_n$ *or* $\mathbb{SS}_m$, *then* s *is in* $\mathbb{U}_n$.

. Firstly, $\mathbb{LPS}_n = \mathbb{N}_n \cup \mathbb{P}_n$ contains all local prominent streaks, and thus all historic moments for $\mathbb{D}_n$. Then, by adapting Lemma 4, we see that given a streak is dominated by $skyline(\mathbb{N}_n)$, then it would not be a historic moment for either $\mathbb{SS}_n$ or $\mathbb{SS}_m$. So $skyline(\mathbb{N}_n) \cup \mathbb{P}_n$ contains all historic moments. Consequently, by adapting Corollary 2,

we see that if a streak is universally dominated by some streak in $\mathbb{P}_n$, it cannot be the historic moment for $\mathbb{SS}_n$ or $\mathbb{SS}_m$. So $\mathbb{U}_n$ contains all historic moments. The lemma follows. ■

Lemma 18. *If a streak s is in $\mathbb{U}_n$, then s can serve as a historic moment of some streak in $\mathbb{SS}_n$ or $\mathbb{SS}_m$.*

. Same as the proof of Lemma 7, with adapting D_n, $\mathcal{SS}_n$, $\mathcal{LPS}_n$, $\mathcal{P}_n$ and $\mathcal{AS}$ to D_n^i, $\mathbb{SS}_n$, $\mathbb{LPS}_n$, $\mathbb{P}_n$ and $\mathbb{AS}$ respectively.

■

Combining the above lemmas, Theorem 4 follows. □

Algorithms 9 and 10 present the MAINTENANCE and LOOK UP procedures of SOIA-MS respectively. They are indeed straightforward adaption of SOIA with minimal changes like computing $\mathcal{LPS}_n^{D_n^i}$ for every data sequence D_n^i and using the multiple sequence version of the definitions instead (e.g., using $\mathbb{P}_n$ instead of $\mathcal{P}_n$).

A.0.3 Performance Study

We compare the performance of BIA-MS and SOIA-MS, using multiple real data sequences. The experimental platform and workload settings are the same as in section 2.5. Among the five real datasets, only D5, the ground motion data sequence from seismic station, has multiple sequences (from multiple stations). So, we construct five real multi-sequence datasets as follows:

Algorithm 9 SOIA-MS Maintenance

1: **procedure** MAINTENANCE($\{\langle v_{n+1}^{D^1}, v_{n+2}^{D^1} \cdots v_m^{D^1} \rangle, \cdots, \langle v_{n+1}^{D^w}, v_{n+2}^{D^w} \cdots v_m^{D^w} \rangle\}$)
2: $\mathscr{N} = \varnothing$;
3: **for** $k = 1$ to $m - n$ **do**
4: **if** $n == 0$ and $k == 1$ **then**
5: $\mathbb{P}_1 = \{(1, 1, v_1^{D^1}), (1, 1, v_1^{D^2}) \cdots (1, 1, v_1^{D^w})\}$;
6: **continue**;
7: $\mathbb{SS}_{n+k} = \varnothing$;
8: **for each** new value $v_{n+k}^{D^j}$ **do**
9: **for each** streak $(i, n + k - 1, v)$ in $\mathcal{SS}_{n+k-1}^{D^j}$ **do**
10: **if** $v_{n+k}^{D^j} \geq v$ **then**
11: Insert $(i, n + k, v)$ to $\mathcal{SS}_{n+k}^{D^j}$;
12: **else if** $|n + k - 1 - i| \geq \sigma$ **then**
13: Insert $(i, n + k - 1, v)$ to Buffer B, along with its data sequence information;
14: **if** no streak in $\mathcal{SS}_{n+k-1}^{D^j}$ has value $v_{n+k}^{D^j}$ **then**
15: **if** all streaks in $\mathcal{SS}_{n+k-1}^{D^j}$ have value $< v_{n+k}^{D^i}$ **then**
16: Insert $(n + k, n + k, v_{n+k}^{D^j})$ to $\mathcal{SS}_{n+k}^{D^j}$;
17: **else**
18: Select the streak $(i, n+k-1, v)$ in $\mathcal{SS}_{n+k-1}^{D^j}$ with $v > v_{n+k}^{D^j}$ and is the smallest;
19: Extend it to be $(i, n + k, v_{n+k}^{D^j})$;
20: Insert it into $\mathcal{SS}_{n+k}^{D^j}$;
21: $\mathbb{SS}_{n+k} = \mathcal{SS}_{n+k}^{D^1} \cup \cdots \cup \mathcal{SS}_{n+k}^{D^w}$
22: $\mathbb{T}$= streaks in Rtree $\cup$ B;
23: Set $\mathbb{P}_{n+k-1}^{\mathbb{T}} = \mathbb{T} \cap \mathbb{P}_{n+k-1}$;
24: Find $\Delta\mathbb{N}$ and $\mathbb{P}_{n+k}$ from $\mathbb{P}_{n+k-1}^{\mathbb{T}}$ and $\mathbb{SS}_{n+k}$;
25: $\mathscr{N} = \mathscr{N} \cup \Delta\mathbb{N}$;
26: Insert streaks in B into Rtree;
27: **for each** streak y in $\mathscr{N}$ **do**
28: Remove streak s from Rtree if $y \succ s$;
29: **for each** streak p in $\mathbb{P}_m$ **do**
30: Remove streak s from Rtree if s is universally dominated by p;

Algorithm 10 SOIA-MS Lookup

1: **procedure** LOOKUP(σ', k')
2: $\mathcal{Z}$ = Get-Top-SS($\mathbb{SS}_n$, k');
3: **for each** streak z in $\mathcal{Z}$ **do**
4: $Q = [|z| \cdot \sigma', +\infty] \times [0, z.i) \times [z.v \cdot \sigma', +\infty]$;
5: $\mathbb{HM}(z)$ = BBS(Rtree, Q), and relevant data sequence information;

1. MS1: KMNB, YHNB
2. MS2: KMNB, YHNB, YULB, TWGB
3. MS3: KMNB, YHNB, YULB, TWGB, TPUB, SSLB
4. MS4: KMNB, YHNB, YULB, TWGB, TPUB, SSLB, NACB, YOJ
5. MS5: KMNB, YHNB, YULB, TWGB, TPUB, SSLB, NACB, YOJ, HK, HK0

where each code (e.g., KMNB) above denotes one station name.

A.0.3.1 Overall Comparison

We first look at the overall performance of SOIA-MS and BIA-MS when the full datasets are available. We evaluate them in terms of their (a) index building time (MAINTENANCE time), (b) query time (LOOK UP time), and (c) index space.

A.0.3.2 Historic Moment Exploration with Data Update

We evaluate the performance of SOIA-MS and BIA-MS for maintaining the index structure online. In this experiment, we regard the first 98% of a dataset

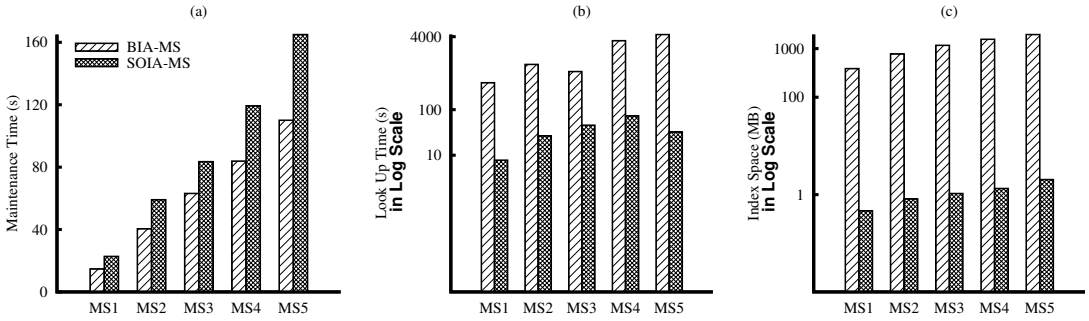
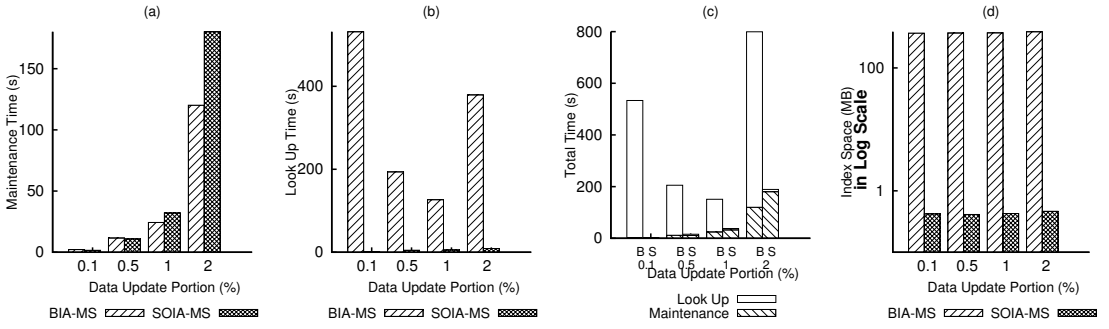


Figure A.1: SOIA-MS vs. BIA-MS

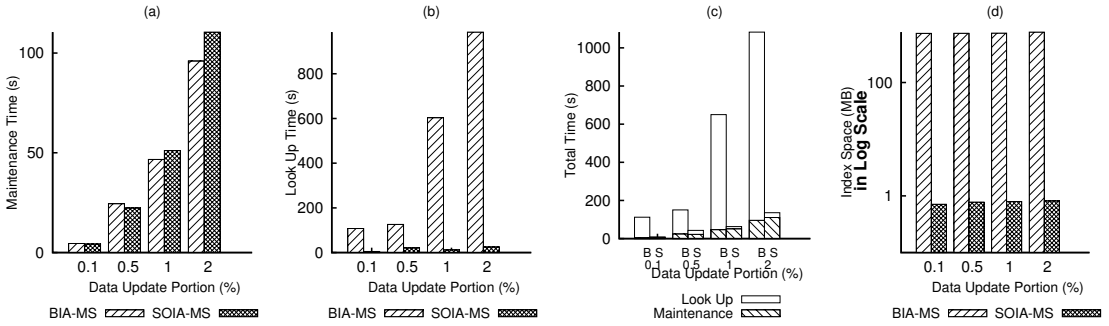
as the initial dataset, i.e., each data sequence in the dataset has 98% as initial, and the index structure of it has been built by maintenance procedure already.

Next, we examine the performance of SOIA-MS and BIA-MS regarding a data append of the last x% of each data sequence in the dataset, where $x = 0.1$, 0.5, 1, and 2. Figures A.2 and A.3 shows the experiment results. While SOIA-MS spends more efforts in maintaining minimality, SOIA-MS is much more efficienct when looking up the historic moments, as SOIA-MS always maintains a much smaller index size.

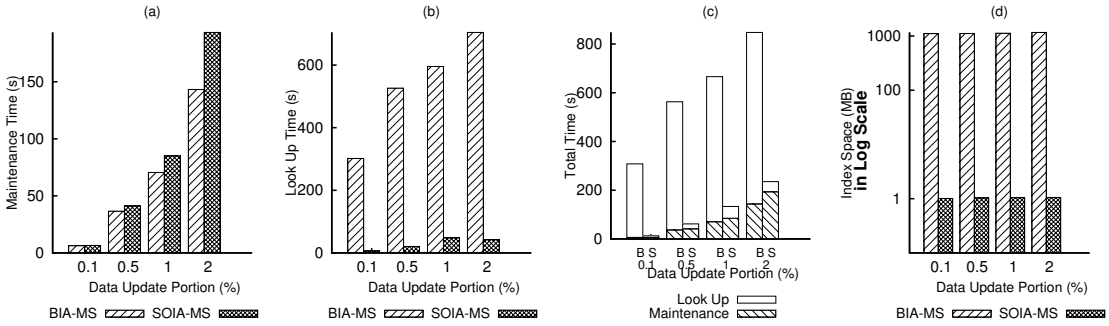
The basic principle of SOIA-MS is similar to SOIA, and the impact of parameters σ and k for SOIA-MS is also similar to SOIA. So here we don't repeat the experiments of parameter sensitivity.



MS1



MS2



MS3

Figure A.2: SOIA-MS vs BIA-MS under data update (MS1 MS2 MS3)

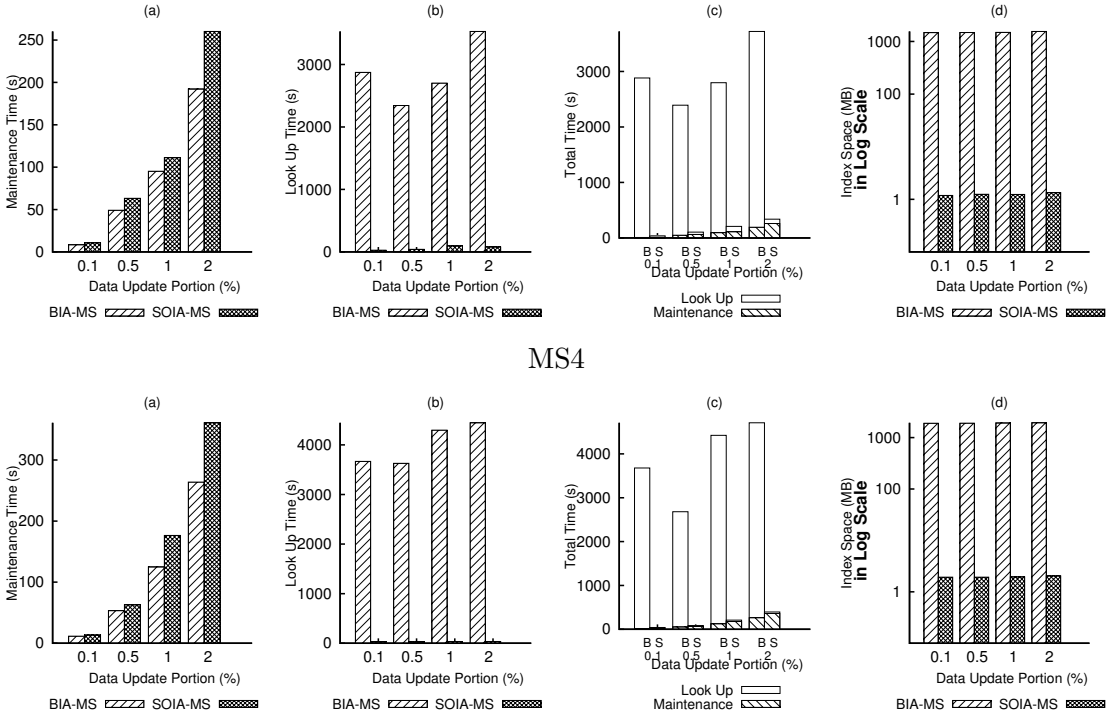


Figure A.3: SOIA-MS vs BIA-MS under data update (MS4 MS5)

Bibliography

[1] Daniel Abadi, Donald Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, C Erwin, Eduardo Galvez, M Hatoun, Anurag Maskey, Alex Rasin, et al. Aurora: a data stream management system. In *SIGMOD Conference*, page 666. Citeseer, 2003.

[2] Foto N Afrati, Paraschos Koutris, Dan Suciu, and Jeffrey D Ullman. Parallel skyline queries. *Theory of Computing Systems*, 57(4):1008–1037, 2015.

[3] Charu C Aggarwal and S Yu Philip. A survey of synopsis construction in data streams. In *Data Streams*, pages 169–207. Springer, 2007.

[4] Charu C Aggarwal and Philip S Yu. On classification of high-cardinality data streams. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 802–813. SIAM, 2010.

[5] Gail M Atkinson and David M Boore. Earthquake ground-motion prediction equations for eastern north america. *Bulletin of the Seismological Society of America*, 96(6):2181–2205, 2006.

[6] Wolf-Tilo Balke, Ulrich Güntzer, and Jason Xin Zheng. Efficient distributed skylining for web information systems. In *International Conference on Extending Database Technology*, pages 256–273. Springer, 2004.

[7] Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Heavy hitters in streams and sliding windows. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[8] Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J Strauss. Space-optimal heavy hitters with strong error bounds. *ACM Transactions on Database Systems (TODS)*, 35(4):26, 2010.

[9] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 421–430. IEEE, 2001.

[10] Eugenio Cesario, Antonio Grillo, Carlo Mastroianni, and Domenico Talia. A sketch-based architecture for mining frequent items and itemsets from distributed data streams. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 245–253. IEEE Computer Society, 2011.

[11] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Automata, languages and programming*, pages 784–784, 2002.

[12] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Random sampling for histogram construction: How much is enough? In *ACM SIGMOD Record*, volume 27, pages 436–447. ACM, 1998.

[13] Aiyou Chen, Yu Jin, Jin Cao, and Li Erran Li. Tracking long duration flows in network traffic. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–5. IEEE, 2010.

[14] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. Skyline with presorting: Theory and optimizations. In *Intelligent Information Processing and Web Mining*, pages 595–604. Springer, 2005.

[15] Sarah Cohen, James T. Hamilton, and Fred Turner. Computational journalism. *Communications of the ACM*, 2011.

[16] Graham Cormode. Count-min sketch. In *Encyclopedia of Database Systems*, pages 511–516. Springer, 2009.

[17] Graham Cormode. Sketch techniques for approximate query processing. *Foundations and Trends in Databases. NOW publishers*, 2011.

[18] Graham Cormode, Theodore Johnson, Flip Korn, Shan Muthukrishnan, Oliver Spatscheck, and Divesh Srivastava. Holistic udafs at streaming speeds. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 35–46. ACM, 2004.

[19] Graham Cormode, Flip Korn, S Muthukrishnan, and Divesh Srivastava. Finding hierarchical heavy hitters in data streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 464–475. VLDB Endowment, 2003.

[20] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[21] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[22] Tiziano De Matteis and Gabriele Mencagli. Keep calm and react with foresight: Strategies for low-latency and energy-efficient elastic data stream processing. In *ACM SIGPLAN Notices*, volume 51, page 13. ACM, 2016.

[23] Cristian Estan and George Varghese. *New directions in traffic measurement and accounting*, volume 32. ACM, 2002.

[24] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*, pages 137–156. Discrete Mathematics and Theoretical Computer Science, 2007.

[25] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[26] Anna C Gilbert, Yannis Kotidis, S Muthukrishnan, and Martin Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Vldb*, volume 1, pages 79–88, 2001.

[27] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[28] Ramesh Govindan, Joseph Hellerstein, Wei Hong, Samuel Madden, Michael Franklin, and Scott Shenker. The sensor network as a database. Technical report, Citeseer, 2002.

[29] Sudipto Guha, Nick Koudas, and Kyuseok Shim. Data-streams and histograms. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 471–475. ACM, 2001.

[30] Naeemul Hassan, Afroza Sultana, You Wu, Gensheng Zhang, Chengkai Li, Jun Yang, and Cong Yu. Data in, fact out: Automated monitoring of facts by factwatcher. In *Proceedings of the 40th International Conference on Very Large Data Bases*. VLDB Endowment, 2014.

[31] Qun Huang, Patrick PC Lee, and Yungang Bao. Sketchlearn: relieving user burdens in approximate measurement with automated statistical inference. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 576–590. ACM, 2018.

[32] Hosagrahar Visvesvaraya Jagadish, Nick Koudas, S Muthukrishnan, Viswanath Poosala, Kenneth C Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. In *VLDB*, volume 98, pages 24–27, 1998.

[33] Xiao Jiang, Chengkai Li, Ping Luo, Min Wang, and Yong Yu. Prominent streak discovery in sequence data. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1280–1288. ACM, 2011.

[34] Christos Kalyvas and Theodoros Tzouramanis. A survey of skyline query processing. *arXiv preprint arXiv:1704.01788*, 2017.

[35] George Kollios, John W Byers, Jeffrey Considine, Marios Hadjieleftheriou, and Feifei Li. Robust aggregation in sensor networks. *IEEE Data Eng. Bull.*, 28(1):26–32, 2005.

[36] George Kollios, John W Byers, Jeffrey Considine, Marios Hadjieleftheriou, and Feifei Li. Robust aggregation in sensor networks. *IEEE Data Eng. Bull.*, 28(1):26–32, 2005.

[37] Donald Kossmann, Frank Ramsak, and Steffen Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 275–286. VLDB Endowment, 2002.

[38] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 234–247. ACM, 2003.

[39] Abhishek Kumar, Minho Sung, Jun Jim Xu, and Jia Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *ACM SIGMETRICS Performance Evaluation Review*, volume 32, pages 177–188. ACM, 2004.

[40] Ashwin Lall, Vyas Sekar, Mitsunori Ogihara, Jun Xu, and Hui Zhang. Data streaming algorithms for estimating entropy of network traffic. In *ACM*

SIGMETRICS Performance Evaluation Review, volume 34, pages 145–156. ACM, 2006.

[41] Ming-Yen Lin, Chao-Wen Yang, and Sue-Chen Hsueh. Efficient computation of group skyline queries on mapreduce. *GSTF Journal on Computing (JoC)*, 5(1), 2016.

[42] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114. ACM, 2016.

[43] Samuel R Madden, Michael J Franklin, Joseph M Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transactions on database systems (TODS)*, 30(1):122–173, 2005.

[44] Nishad Manerikar and Themis Palpanas. Frequent items in streaming data: An experimental evaluation of the state-of-the-art. *Data & Knowledge Engineering*, 68(4):415–430, 2009.

[45] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 346–357. Elsevier, 2002.

[46] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *International Conference on Database Theory*, pages 398–412. Springer, 2005.

[47] Ahmad Mustafa, Ahsanul Haque, Latifur Khan, Michael Baron, and Bhavani Thuraisingham. Evolving stream classification using change detection.

In *2014 International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 154–162. IEEE, 2014.

[48] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An optimal and progressive algorithm for skyline queries. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 467–478. ACM, 2003.

[49] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems*, 2005.

[50] Loïc Paulevé, Hervé Jégou, and Laurent Amsaleg. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348–1358, 2010.

[51] Guillaume Pitel and Geoffroy Fouquier. Count-min-log sketch: Approximately counting with approximate counters. *arXiv preprint arXiv:1502.04885*, 2015.

[52] Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1449–1463. ACM, 2016.

[53] Robert Schweller, Ashish Gupta, Elliot Parsons, and Yan Chen. Reversible sketches for efficient and accurate change detection over network data streams. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 207–212. ACM, 2004.

[54] Vyas Sekar, Nick G Duffield, Oliver Spatscheck, Jacobus E van der Merwe, and Hui Zhang. Lads: Large-scale automated ddos detection system. In *USENIX Annual Technical Conference, General Track*, pages 171–184, 2006.

[55] Michael Shekelyan, Anton Dignös, and Johann Gamper. Digithist: a histogram-based data summary with tight error bounds. *Proceedings of the VLDB Endowment*, 10(11):1514–1525, 2017.

[56] David C Steere, Antonio Baptista, Dylan McNamee, Calton Pu, and Jonathan Walpole. Research challenges in environmental observation and forecasting systems. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 292–299. ACM, 2000.

[57] Afroza Sultana, Naeemul Hassan, Chengkai Li, Jun Yang, and Cong Yu. Incremental discovery of prominent situational facts. In *Proceedings of the IEEE 30th International Conference on Data Engineering*. IEEE, 2014.

[58] Kian-Lee Tan, Pin-Kwang Eng, Beng Chin Ooi, et al. Efficient progressive skyline computation. In *VLDB*, volume 1, pages 301–310, 2001.

[59] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 563–576. ACM, 2009.

[60] Daniel Ting. Data sketches for disaggregated subset sum and frequent item estimation. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1129–1140. ACM, 2018.

[61] Chuyuan Wei and Yongzhen Li. Design of energy consumption monitoring and energy-saving management system of intelligent building based on the internet of things. In *Electronics, Communications and Control (ICECC), 2011 International Conference on*, pages 3650–3652. IEEE, 2011.

[62] Kyu-Young Whang, Brad T Vander-Zanden, and Howard M Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems (TODS)*, 15(2):208–229, 1990.

[63] Wikipedia. 2016 taiwan earthquake — wikipedia, the free encyclopedia, 2017. [Online; accessed 27-February-2017].

[64] You Wu, Pankaj K Agarwal, Chengkai Li, Jun Yang, and Cong Yu. On one of the few objects. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495. ACM, 2012.

[65] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 561–575. ACM, 2018.

[66] Tong Yang, Yang Zhou, Hao Jin, Shigang Chen, and Xiaoming Li. Pyramid sketch: a sketch framework for frequency estimation of data streams. *Proceedings of the VLDB Endowment*, 10(11):1442–1453, 2017.

[67] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *NSDI*, volume 13, pages 29–42, 2013.

[68] Yidong Yuan, Xuemin Lin, Qing Liu, Wei Wang, Jeffrey Xu Yu, and Qing Zhang. Efficient computation of the skyline cube. In *Proceedings of the 31st International Conference on Very Large Data Bases*. VLDB Endowment, 2005.

[69] Gensheng Zhang, Xiao Jiang, Ping Luo, Min Wang, and Chengkai Li. Discovering general prominent streaks in sequence data. *ACM Transactions on Knowledge Discovery from Data*, 8(2):9:1–9:37, June 2014.

[70] Huanchen Zhang, Hyeontaek Lim, Viktor Leis, David G Andersen, Michael Kaminsky, Kimberly Keeton, and Andrew Pavlo. Surf: Practical range query filtering with fast succinct tries. In *Proceedings of the 2018 International Conference on Management of Data*, pages 323–336. ACM, 2018.

[71] Haiquan Chuck Zhao, Ashwin Lall, Mitsunori Ogihara, Oliver Spatscheck, Jia Wang, and Jun Xu. A data streaming algorithm for estimating entropies of od flows. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 279–290. ACM, 2007.