Discovering Newsworthy Themes from Sequenced Data: A Step Towards Computational Journalism

Qi Fan, Yuchen Li, Dongxiang Zhang, and Kian-Lee Tan, *Member, IEEE*

Abstract—Automatic discovery of newsworthy themes from sequenced data can relieve journalists from manually poring over a large amount of data in order to find interesting news. In this paper, we propose a novel k-Sketch query that aims to find k striking streaks to best summarize a subject. Our scoring function takes into account streak strikingness and streak coverage at the same time. We study the k-Sketch query processing in both offline and online scenarios, and propose various streak-level pruning techniques to find striking candidates. Among those candidates, we then develop approximate methods to discover the k most representative streaks with theoretical bounds. We conduct experiments on four real datasets, and the results demonstrate the efficiency and effectiveness of our proposed algorithms: the running time achieves up to 500 times speedup and the quality of the generated summaries is endorsed by the anonymous users from Amazon Mechanical Turk.

Index Terms—Computational journalism, news theme discovery, sequenced data, approximate algorithms

1 Introduction

Today's journalists must pore over large amounts of data to discover attention-seizing facts as *news themes*. While such a task has traditionally been done manually, there is an increasing reliance on computational technology [1], [2], [3] to reduce human labor and intervention to a minimum. Recently, Zhang et al. have proposed a new type of news theme, named *prominent streaks* [4]. A streak corresponds to a set of consecutive sequenced events belonging to the same subject, such as the latest ten games a player participated. The prominent streaks are defined as the skylines among all streaks of a subject, and can be used to represent the history of the subject. In [4], each prominent streak is viewed as a striking news theme since it is outstanding (i.e., non-dominated) in a subject's history.

However, there are two major drawbacks that limit its usability in real applications. First, the prominent streaks generated by [4] may not be striking enough because they are derived from the historical data of a single subject without comparing to other subjects. For example, "Steve Nash has scored 15+ points in consecutive 10 games" is a prominent streak for "Steve Nash", but it is not striking given the fact that there are more than 90 players with better performance.[1] Second, the number of the prominent streaks can be overwhelming. Since prominent streaks in [4] are skylines, a subject with n historical events may generate at most n streaks that are not dominated (i.e., prominent streaks). Therefore, there calls for a new method to automatically select a limited number of striking streaks which best summarize a subject's history.

In this paper, we solve the problem of effectively and efficiently summarizing a subject's history by proposing a *k-Sketch* query. The k-Sketch query is based on a novel rank-aware streak, named *ranked-streak* and it selects k ranked-streaks which best summarize a subject's history. Compared with previous works, the ranked-streak is able to capture the strikingness of a streak which is very newsworthy as evidenced in the following news excerpts:

(1) (26 Feb 2003) With 32 points, Kobe Bryant saw his 40+ scoring streak end at **nine** games, tied with Michael Jordan for the *fourth* place on the all-time list.[2]

(2) (14 April 2014) Stephen Curry has made 602 3-pointer attempts from beyond the arc, ... are the *10th* most in NBA history in a season (*82 games*).[3]

(3) (28 May 2015) Stocks gained for the *seventh consecutive day* on Wednesday as the benchmark moved close to the 5,000 mark for *the first* time in seven years.[4]

(4) (9 Jun 2014) Delhi has been witnessing a spell of hot weather over the *past month*, with temperature

• Q. Fan is with the NUS Graduate School for Integrative Sciences and Engineering (NGS), National University of Singapore (NUS), Singapore 119077. E-mail: fan.qi@u.nus.edu.
• Y. Li is with the School of Computing, National University of Singapore, Singapore 119077. E-mail: liyuchenmike@gmail.com.
• D. Zhang is with the University of Electronic Science and Technology of China, Sichuan Sheng 610051, China. E-mail: zhangdo@uestc.edu.cn.
• K.-L. Tan is with the School of Computing, NGS, National University of Singapore, Singapore 119077. E-mail: tankl@comp.nus.edu.sg.

*Manuscript received 26 Mar. 2016; revised 21 Jan. 2017; accepted 11 Mar. 2017. Date of publication 21 Mar. 2017; date of current version 1 June 2017. Recommended for acceptance by J. Tang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2017.2685587*

1. http://www.sporcle.com/games/nbadarinh/nba-all-players-with-10-consecutive-20-point-games-90-11
2. http://www.nba.com/features/kobe_40plus_030221.html
3. http://www.cbssports.com/nba/eye-on-basketball/24525914/stephen-curry-makes-history-with-consecutive-seasons-of-250-3s
4. http://www.zacks.com/stock/news/176469/china-stock-roundup-ctrip-buys-elong-stake-trina-solar-beats-estimates

TABLE 1
Indicators of Ranked-Streaks

E.g. & Subject	Aggregate Function	Streak Length	Rank
1. Kobe	min (points)	9 straight games	4
2. Stephen	sum (shot attempts)	82 games	10
3. Stock Index	count (gains)	7 consecutive days	1
4. Delhi	average (degree)	past months (30 days)	1
5. Pelican Point	max (raindrops)	12 months	9

hovering around 45 degrees Celsius, ... *highest* ever since 1952.[5]

(5) (22 Jul 2011) Pelican Point recorded a maximum rainfall of 0.32 inches for *12 months*, making it the *9th driest* places on earth.[6]

In the above examples, each news theme is a *ranked-streak* which consists of five components: (1) a subject (e.g., Kobe Bryant, Stocks, Delhi), (2) a streak length (e.g., nine straight games, seventh consecutive days, past month), (3) an aggregate function on an attribute (e.g., minimum points, count of gains, average of degrees), (4) a rank (e.g., fourth, first time, highest), and (5) a historical dataset (e.g., all time list, seven years, since 1952). The indicators (1)-(4) are summarized in Table 1. Based on the *ranked-streaks*, our k-Sketch query leverages a novel scoring function that chooses the best k ranked-streaks to summarize a subject's history. Our scoring function considers two aspects: (1) we prefer the ranked-streaks that covers as many events as possible to represent a subject's history, and (2) we prefer the ranked-streaks that have better ranks[7] as they indicate more strikingness. Our objective is to process the k-Sketch query for each subject in the domain.

We study the k-Sketch query processing under both offline and online scenarios. In the offline scenario, our objective is to efficiently discover the sketches for each subject from historical data. A challenging prerequisite is to generate the ranked-streaks. The brute-force approach is to enumerate all streaks to generate ranks. Such an approach has a quadratic complexity wrt. the number of events, which is not scalable even for moderate data size. By leveraging the subadditivity among the upper bounds of streaks, we design two effective pruning techniques to facilitate efficient ranked-streak generation. Next, we notice that generating exact sketches from ranked-streaks is expensive. Thus, we design an efficient $(1-1/e)$-approximate algorithm by exploiting submodularity of the k-Sketch query.

In the online scenario, fresh events are continuously fed into the system and our goal is to maintain the sketches for each subject up-to-date. When a new event about subject s arrives, many ranked-streaks of various lengths can be derived. For each derived ranked-streak, not only the sketch of s but also the sketches of other subjects may be affected. Dealing with such a complex updating pattern is non-trivial. The naive scheme, in the worst case, is to examine every ranked-streak for every subject to maintain the $(1-1/e)$ approximation, which is prohibitive in the online setting. To efficiently support the update while maintaining the quality of sketches, we propose a $1/8$-approximate solution which only examines $2k$ ranked-streaks for each subject whose sketch is affected.

Our contributions are hereby summarized as follows:

- We study the problem of automatic summarization of a subject's history. We utilize the ranked-streak that is a common news theme in real-life reports but has not been addressed in previous works. We formulate the summarization problem as a k-Sketch query under a novel scoring function that considers both strikingness and coverage.
- We study the k-Sketch query processing in both online and offline scenarios. In the offline scenario, we propose two novel pruning techniques to efficiently generate ranked-streaks. Then we design a $(1-1/e)$-approximate algorithm to compute the sketches for each subject. In the online scenario, we propose a $1/8$-approximate algorithm to efficiently support the complex updating patterns as new event arrives.
- We conduct extensive experiments with four real datasets to evaluate the effectiveness and the efficiency of our proposed algorithms. In the offline scenario, our solution is three orders of magnitude faster than baseline algorithms. While in the online scenario, our solution achieves up to 500x speedup. In addition, we also perform an anonymous user study via Amazon Mechanical Turk platform, which validates the effectiveness of our k-Sketch query.

The rest of the paper is organized as follows. In Section 2, we summarize the related work. Section 3 formulates the k-Sketch query. Section 4 presents the algorithms for processing the k-Sketch query in the offline scenario. Section 5 describes the algorithms for maintaining k-Sketches in the online scenario. In Section 6, comprehensive experimental studies on both the efficiency and the effectiveness of our algorithms are conducted. Section 7 discusses the extension of our methods. Finally, Section 8 concludes our paper.

2 Related Work

In this section, we briefly review the following closely related areas to our k-Sketch query: news discovery in computational journalism, frequent episode mining from sequenced events, top-k diversity query and event detection in information retrieval.

2.1 Computational Journalism

An important aspect of computational journalism is to leverage computational technology to discover striking news themes. Previous works on automatic news theme generation belong to two categories: *dimension-oriented* approach and *subject-oriented* approach. Dimension-oriented approaches aim to select appropriate dimensions to make an event interesting. Representative works include the *situational facts* [1] and the *one-of-the-few facts* [2]. On the other hand, subject-oriented approaches aim to summarize subjects' histories from their historical events, such as the *prominent streaks* [4]. Our proposed solution falls into the subject-oriented category.

5. http://www.dnaindia.com/delhi/report-delhi-records-highest-temperature-in-62-years-1994332
6. http://www.livescience.com/30627-10-driest-places-on-earth.html
7. We consider a ranked-streak with rank i to be more attractive than j when $i < j$ and the other fields are the same.

TABLE 2
Examples of Different News Themes

Method	Example news theme
Situational facts [1]	Damon Stoudamire scored 54 points on January 14, 2005. It is the highest score in history made by any Trail Blazer.
One-of-the-τ facts [2]	Jordan, Chamberlain, James, Baylor and Pettit are the five players with highest points and rebounds in NBA history.
Prominent streaks [4]	1. Kobe scored 40+ in 9 straight games, first in his career! 2. Kobe scored 50+ in 4 straight games, first in his career! 3 . . .
k-Sketch	1. Kobe scored 40+ in 9 straight games, ranked 4th in NBA history! 2. Kobe scored 50+ in 4 straight games, ranked 1st in NBA history. 3 . . .

2.1.1 Situational Facts and One-of-the-Few Facts

Situational Facts [1]: it finds for a given event, the best constraint-measure pair that makes the event unique (i.e., not dominated by others). An example of the *situational fact* is listed in Table 2 and is extracted from the NBA game events where the measure dimensions are “points, assists, and rebounds” and the constraint dimensions are “team, result, and date”. The *situational fact* of the event in Table 2 is the constraint-measure pair $\langle$team = Blazer, points$\rangle$, since under this situation, this event is a skyline among all events (i.e., no other events matching the constraint “team = Blazer” contain an even higher “point”.).

One-of-the-Few Facts [2]: it finds the dimensions under which no more than τ events are in the k-skyband (i.e., not dominated by k events and k is as small as possible). An example of the *one-of-the-few fact* is listed in Table 2. In the example, when $\tau = 5$, two dimensions are selected (i.e., points and rebounds). Under these two dimensions, five players are the skylines (i.e., one-skyband), thus each of them is a *one-of-the-5* player.

As demonstrated, “situational facts” and “one-of-the-few facts” are dimension-oriented since they attempt to generate news themes by selecting dimensions.

2.1.2 Prominent Streaks

Zhang et al. [4] proposed a subject-oriented approach to generate news themes by discovering *prominent streaks*. In [4], a streak is modeled as a pair of the streak duration and the minimum value of all events in the streak. For example, as shown in Table 2, a streak of “Kobe” may be $\langle$9 consecutive games, minimum points of 40$\rangle$. The objective of [4] is to discover all the non-dominated streaks (i.e., prominent streaks) where the dominance is defined among streaks of the same subject. Despite being the same subject-oriented approach, our k-Sketch query differs from [4] in two aspects. First, we look at the *global prominence among all* subjects (i.e., rank in NBA history) rather than local prominence within one subject (i.e., non-dominance in one’s career). Second, our model provides the best k ranked-streaks for each subject, whereas [4] returns a set of skylines which could be potentially large.

2.2 Frequent Episode Mining

In sequenced data mining, an episode [5], [6], [7], [8] is defined as a collection of time sequenced events which occur together within a time window. The uniqueness of an episode is determined by the containing events. The objective of frequent episode mining is to discover episodes whose occurrences exceed a threshold. Our k-Sketch query differs from the episode mining in two major aspects. First, episodes are associated with categorical values thus they can be grouped to count the occurrences. On the other hand, our ranked-streaks are defined with numerical values, making it inappropriate to be grouped. Second, the episodes are selected based on the occurrences which do not contain the *rank* information, whereas our k-Sketch query explicitly provides the rank of selected streaks. As such, episode mining techniques cannot support k-Sketch query.

2.3 Top-k Diversity Query

Top-k diversity queries [9], [10], [11], [12] aim to find a subset of objects to maximize a scoring function. The scoring function normally penalizes subsets with similar elements. Our k-Sketch query has two important distinctions. First, the inputs to top-k diversity queries are known in advance, whereas in k-Sketch query, the ranks of streaks needs to be derived. Second, existing methods for online diversity queries [10], [11], [12] only study the update on a single result set when a new event arrives. However, our online sketch maintenance incurs the problem of multiple sketch updates for each new event. Such a complex update pattern has not been studied yet.

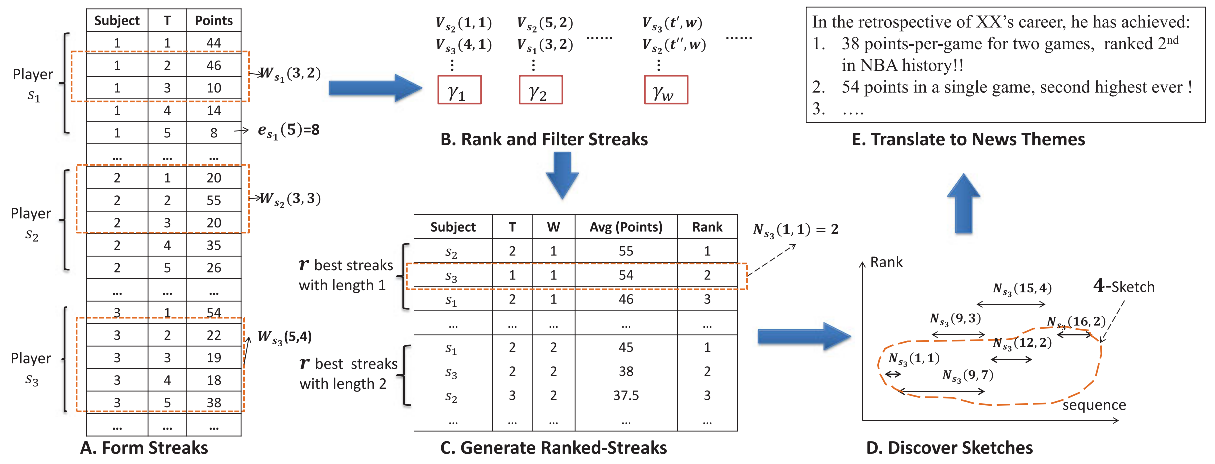
2.4 Event Detection and Tracking

In information retrieval, event detection and tracking aim to extract and organize new events from various media sources such as text streams [13], [14], social media streams [15] and web articles [16]. Despite the usefulness of these works, they differ from our k-Sketch query as they focuses on the detection of a single event, whereas k-Sketch aims to summarize a subject’s history. Therefore, the abovementioned techniques cannot be directly applied.

3 Problem Formulation

Let S denote a set of subjects of potential interests to journalists. For example, S can refer to all the players or teams in the NBA application. Let $e_s(t)$ denote an event about subject s, where t denotes its timestamp or sequence ID. For example, an event can refer to an NBA game a player participated on a certain day. Note that we maintain a sequence ID for each subject that is automatically incremented. It is possible that the events of different subjects may have the same sequence ID. Consecutive events of the same subject can be grouped as a *streak*:

Definition 1 (Streak [4]). *Let w be a streak length, a streak $W_s(t, w)$ refers to w consecutive events of subject s ending at sequence t, i.e., $W_s(t, w) = \{e_s(t - w + 1), \ldots, e_s(t)\}$.*



Subject	T	Points
1	1	44
1	2	46
1	3	10
1	4	14
1	5	8
...	...	...
2	1	20
2	2	55
2	3	20
2	4	35
2	5	26
...	...	...
3	1	54
3	2	22
3	3	19
3	4	18
3	5	38
...	...	...

Subject	T	W	Avg (Points)	Rank
s_2	2	1	55	1
s_3	1	1	54	2
s_1	2	1	46	3
...	...	...	...	...
s_1	2	2	45	1
s_3	2	2	38	2
s_2	3	2	37.5	3
...	...	...	...	...

In the retrospective of XX's career, he has achieved:
1. 38 points-per-game for two games, ranked 2nd in NBA history!!
2. 54 points in a single game, second highest ever !
3. ….

Fig. 1. An illustration of k-Sketch query processing. (A): Various streaks are formed based on events' sequence IDs. (B): Streaks with rank greater than $\texttt{r}$ are filtered. (C): Ranked-streaks for each streak length. (D): k-sketches are discovered for each subject from their ranked-streaks. (E): Newsworthy summary of a subject can be generated from its k-sketches.

If a subject s has $|\mathbb{H}_s|$ events, then there are $\binom{|\mathbb{H}_s|}{2}$ possible streaks.[8] Given an aggregate function f, events in a streak can be aggregated to a numerical value $\overline{v}$ as

$$V_{s_1}(t,w) = f(W_s(t,w)) = f(e_s(t-w+1),\dots,e_s(t)).$$

Common aggregate functions include *sum*, *avg*, *count*, *min* and *max*. In this paper, we only consider a single aggregate function. Multiple aggregate functions can be simply processed independently.

Example 1. Fig. 1A illustrates examples of *streak*s of three NBA players. Each event records the points scored by a player in a game. In the figure, the streak $W_{s_1}(3,2)$ refers to two consecutive events about player s_1 ending at $t = 3$, i.e., $W_{s_1}(3,2) = \{e_{s_1}(2), e_{s_1}(3)\}$. Given an aggregate function $f = avg(points)$, we yield $V_{s_1}(3,2) = (46+10)/2 = 28$.

With the aggregated value $\overline{v}$, we can derive the *rank* of a streak to measure its strikingness. For instance, "*The total points Kobe has scored is 32,482*" can be transformed into a rank-aware representation: "*Kobe moved into third place on the NBA's all-time scoring list*", in which the rank is 3. Let W_s be a length-w streak, and $\gamma_w(V_s)$ be the rank of W_s by comparing it with all other length-w streaks on their aggregate values. In general, a streak is striking if its rank is smaller than a predefined threshold (i.e., top-$\texttt{r}$). This leads to our definition of *Ranked-Streak*:

Definition 2 (Ranked-Streak). *A streak $W_s(t,w)$ can be transformed to a ranked-streak, denoted by $N_s(t,w)$, if its rank $\gamma_w(V_s(t,w)) \leq \texttt{r}$, where $\texttt{r}$ is a user-defined threshold.*

Example 2. In step (B) of Fig. 1, we group the streaks based on their lengths. Each streak is associated with a rank value γ_w. If the rank is greater than the threshold $\texttt{r}$, the associated streak is pruned. Otherwise, it is considered as a ranked-streak. In step (C) of Fig. 1, we present the ranked-streaks in the tabular format. Each ranked-streak contains the rank computed from step (B). For instance, among all ranked-streaks with length 1, $N_{s_3}(1,1)$ (with value 54) is ranked 2nd because its aggregated value is smaller than that of another streak $N_{s_2}(2,1)$ (with value 55).

8. This number is derived by constructing a streak using start and end sequence IDs.

Let $\mathbb{N}_s$ be the set of ranked-streaks of a subject s. Since there are at most $\texttt{r}$ ranked-streaks for each possible streak lengths, the size of $\mathbb{N}_s$ can be as large as $\texttt{r}|\mathbb{H}_s|$ which is still overwhelming for summarizing the subject's history. To control the output size while maintaining the quality of the summary, we aim to find a subset of k ranked-streaks from $\mathbb{N}_s$ which best summarize the history of s. We call such a subset a *k-Sketch*.

To measure the quality of the selected ranked-streaks, we define a scoring function $g(\cdot)$. The design of g gives rise to two concerns. First, g prefers ranked-streaks covering as many of the subject's historical events as possible. This is because a higher coverage indicates fewer missing historical events in the summary. Second, g needs to assign a higher score to the ranked-streaks with better ranks. This is because a better rank indicates better strikingness which implies that the news themes generated would be more eye-catching. For instance, we often care more about who the top scorer in NBA history is rather than who is ranked 50th.

To address these two concerns, we define g as follows: let $\mathbb{X}_s$ be a set of ranked-streaks about subject s (i.e., $\mathbb{X}_s \subseteq \mathbb{N}_s$), the score of $\mathbb{X}_s$ is

$$g(\mathbb{X}_s) = \alpha C(\mathbb{X}_s) + (1-\alpha)R(\mathbb{X}_s), \alpha \in [0,1], \tag{1}$$

where $C(\mathbb{X}_s)$ is the ratio between the number of distinct events covered by $\mathbb{X}_s$ over the total number of events about subject s. In this way, ranked-streaks with a poor coverage contribute to a low score. $R(\mathbb{X}_s) = \frac{1}{|\mathbb{X}_s|}\sum_{X_s \in \mathbb{X}_s} \frac{p - X_s.\gamma}{p}$ is the strikingness of $\mathbb{X}_s$. Any ranked-streak in $\mathbb{X}_s$ changing to a better rank increases $R(\mathbb{X}_s)$. The values of $C(\mathbb{X}_s)$ and $R(\mathbb{X}_s)$ are normalized to $[0,1]$. α is an adjustable coefficient to balance the weights between $C(\mathbb{X}_s)$ and $R(\mathbb{X}_s)$. If α is high, it means users are more interested in finding ranked-streaks that cover most of the subject's history. If α is low, it indicates that users prefer more striking ranked-streaks.

With Eqn. (1), we then define the *k-Sketch Query* as the following optimization problem:

Definition 3 (k-Sketch Query). *Given a parameter k, the k-Sketch Query aims to, for each subject s, find a subset of streaks SK_s to maximize $g(SK_s)$ subject to (1) Cardinality*

TABLE 3
Notations Used in This Paper

Notation	Meaning
S	set of subjects
$\mathbb{H}_s$	set of events associated with subject s
$W_s(t,w)$	length-w streak of s ending at t
$V_s(t,w)$	aggregated value of all events in $W_s(t,w)$
$\mathtt{r}$	minimum rank requirement for ranked-streaks
$N_s(t,w)$	the ranked-streak derived from $W_s(t,w)$
$\mathbb{N}_s$	set of ranked-streaks associated with subject s
$WI(w)$	top-$\mathtt{r}$ ranked-streaks of length w
$\beta(w)$	lower bound of $WI(w)$
SK_s	sketch for subject s
J_s	visiting-streak bound for subject s
M_s	unseen-streak bound for subject s
P_s	online-streak bound for subject s
$g(\cdot)$	scoring function on a set of ranked-streaks
α	user's preference between cover and rank in $g(\cdot)$

Constraint: $|SK_s| = k$ *and (2) Rank Constraint:* $SK_s \subseteq \mathbb{N}_s$, *i.e.,* $\forall W_s(t,w) \in SK_s, \gamma_w(V_s(t,w)) \leq \mathtt{r}$.

Note that k and $\mathtt{r}$ are both user-defined and not directly correlated. In practice, $\mathtt{r}$ requires user's domain knowledge in determining the strikingness of streaks. k is based on the user's demand in analyzing a subject's history.

Example 3. In step (D) of Fig. 1, we show a collection of ranked-streaks and a k-Sketch derived from them. The y-axis is the rank and the x-axis represents the complete sequence of events of a subject. Each ranked-streak is represented by a line segment, covering consecutive events. When $k = 4$, four of the ranked-streaks are selected as the four-Sketch.

Before we move on to the algorithmic part, we first list the frequently used notations in Table 3. For ease of presentation, we present our techniques using *avg* as the default aggregate function. Extending our techniques to other aggregate functions is addressed in Section 7.

4 OFFLINE k-SKETCH QUERY PROCESSING

In the offline scenario, the input is a set of events of all subjects and the output is a k-sketch for each subject s, denoted by SK_s.

Since $\mathbb{N}_s$ is unknown, we need to compute it before optimizing $g(SK_s)$. This naturally leads to a two-step approach as outlined in Fig. 1: ranked-streak generation and k-sketch discovery. In ranked-streak generation, $\mathbb{N}_s$ are generated wrt. the *Rank Constraint*. In subsequent k-sketch discovery, SK_s can be selected from $\mathbb{N}_s$. In this way, only the *Cardinality Constraint* needs to be considered.

4.1 Ranked-Streak Generation

Generating ranked-streak for each subject is computationally expensive. In order to generate accurate ranks for selecting ranked-streaks, a brute-force approach needs to evaluate all the streaks with every possible length. Since there could be $\binom{\mathbb{H}_s}{2}$ streaks associated with subject s, the total time complexity for the brute-force approach is $O(\sum_{s\in S} |\mathbb{H}_s|^2)$. Such a complexity makes the solution infeasible even for moderate sized datasets.

To improve the efficiency, we observe that it is unnecessary to compute all the streaks to generate $\mathbb{N}_s$. The intuition behind is that the upper bound value of streaks with longer lengths can be estimated from those with shorter lengths. This means that we can compute streaks with increasing lengths, and as the shorter streaks are computed, the longer streaks not in $\mathbb{N}_s$ can be pruned. To realize such an intuition, we design the ranked-streak generation algorithm by adopting two novel streak-based pruning methods which exploit the *subadditivity* property among streaks with different lengths.

4.1.1 Overview of Streak Pruning

For each subject, its streaks can be grouped by lengths. Our ranked-streak generation algorithm gradually evaluates a subject's streak from a shorter length to a longer length. To support efficient pruning, we define two concepts, namely *visiting-streak bound* (J_s) and *unseen-streak bound* (M_s). In particular, $J_s(w)$ is the upper bound of all the streaks about subject s with length w, i.e., $\forall w, J_s(w) \geq \max\{V_s(t,w) | t > w\}$. $M_s(w)$ is the upper bound of all the streaks about subject s with a length larger than w, i.e., $M_s(w) \geq \max\{J_s(w_1) | w_1 > w\}$. These two bounds will be used for streak pruning and we will present how to derive these two bounds in Sections 4.1.2 and 4.1.3.

Algorithm 1. Ranked-Streak Generation Overview

```
1: $WI() \leftarrow \{\}$ // top-r streaks for each length
2: $\beta \leftarrow \{\}$ // smallest scores in $WI$ for each length
3: $Q \leftarrow \{(s, 1, +\infty) | s \in S\}$
4: while $(s, w, q) \leftarrow Q.pop()$ do
5:     compute $J_s(w)$
6:     if $J_s(w) > \beta(w)$ then
7:         for $t \in w \ldots |\mathbb{H}_s|$ do
8:             Update $WI(w)$, $\beta(w)$, and $J_s(w)$
9:         end for
10:    end if
11:    compute $M_s(w)$ whose value relies on $J_s(w)$
12:    if $M_s(w) > \min\{\beta(w') | w < w' \leq |\mathbb{H}_s|\}$ then
13:        $q \leftarrow M_s(w)$
14:        $Q.push(s, w+1, q)$
15:    end if
16: end while
17: return $WI$
```

The overview of ranked-streak generation algorithm is presented in Algorithm 1. We maintain two global structures WI and β (Lines 1-2). For each length w, $WI(w)$ stores the top-$\mathtt{r}$ streaks with length w among all the subjects and $\beta(w)$ is the $\mathtt{r}$th value among the streaks in $WI(w)$. In other words, $\beta(w)$ serves as a lower bound value. A streak with length w is a ranked-streak only if its aggregate value is larger than $\beta(w)$. A priority queue Q (Line 3) is used to provide an access order among subjects. Each element in the queue is a triple (s, w, q), where s is a subject, w indicates the next streak length of s to evaluate, and q denotes the priority. We use the unseen-streak bound (i.e., $M_s(w)$) as the priority during every iteration (Lines 13-14). Intuitively, a subject with higher $M_s(w)$ is more likely to spawn new streaks that can increase β and benefit the subsequent pruning. During initialization, we insert, for each subject, an entry with length 1 and priority infinity into Q.

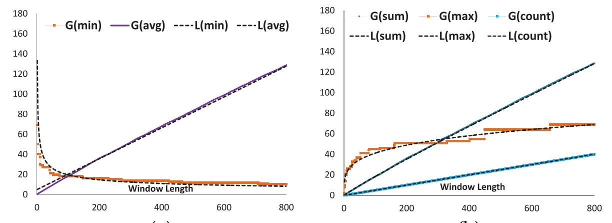


Fig. 2. Illustration of fitting function L on various aggregation functions; solid lines represent G while dotted lines represent L. (a) Fitting on min and average and (b) fitting on max, sum, and count.

In each iteration, we pop the streak with the highest priority (Line 4). Then, we compute the visiting-streak bound $J_s(w)$ for the subject and determine whether all the length-w streak about subject s can be pruned (Lines 5-6). If the bound $J_s(w)$ is no greater than $\beta(w)$, then all these streaks can be ignored. Otherwise, all the length-w streaks of s are computed to update $WI(w)$ and $\beta(w)$ accordingly (Lines 7-9). In the next step, we estimate the unseen-streak bound M_s and compare it with the minimum $\beta(w')$ for all $w' > w$. If M_s is smaller, then we would not find a better streak and all the streaks about subject s with lengths larger than w can be pruned (Lines 11-12). Otherwise, we set the priority of M_s to q, and push the triple $(s, w+1, q)$ back to Q (Lines 13-14). The algorithm terminates when Q becomes empty.

4.1.2 *Visiting-Streak Pruning*

In Algorithm 1, we need to compute the visiting-streak bound $J_s(w)$ to facilitate pruning all length-w streaks associated with subject s. Our idea is that suppose we have successfully derived the bounds for streaks with smaller lengths, we can use them to estimate the bounds of larger streaks. We formulate it as the subadditivity property and use avg[9] as the aggregate function for illustration:

Theorem 1 (Subadditivity (for avg)).

$$\max_t\{V_s(t,w)\} \le \frac{w_iJ_s(w_i) + (w-w_i)J_s(w-w_i)}{w}, \forall w_i \in (0,w). \quad (2)$$

Proof. Given any streak $W = W_s(t,w)$ and a value $w_i \in (0,w)$, we can split the streak into two non-overlapping sub-streaks with lengths w_i and $w - w_i$, i.e., $W_{s1} = W_s(t, w_i)$ and $W_{s2} = W_s(t-w_i, w-w_i)$. Due to the non-overlapping property of W_{s1} and W_{s2}, we have $wV_s = w_iV_{s1} + (w-w_i)V_{s2}$. Since $J_s(w_i)$ and $J_s(w-w_i)$ are two visiting-streak bounds, then $V_{s1} \le J_s(w_i)$ and $V_{s2} \le J_s(w-w_i)$ must hold for any t. It then follows that, for any t, $wV_s \le w_iJ_s(w_i) + (w-w_i)J_s(w-w_i)$, which completes the proof. □

With Theorem 1, we can estimate $J_s(w)$ by any pair $J_s(w_i)$ and $J_s(w-w_i)$, $\forall w_i < w$. Let w^* correspond to the tightest $J_s(w)$, then w^* can be formulated as

$$w^* = \operatorname*{argmin}_{w_i \in (0,w)} \frac{w_iJ_s(w_i) + (w-w_i)J_s(w-w_i)}{w}. \quad (3)$$

9. Although here we demonstrate the bound using "avg", the properties and bounds also hold for other aggregate functions. Corresponding properties and bounds for other aggregate functions are listed in Section 7.

A naive solution to compute w^* would enumerate every possible w_i. However, such a solution has a worst time complexity of $O(|\mathbb{H}_s|)$ for subject s. To quickly compute w^* without enumerating all w_i, we apply a continuous relaxation to Eqn. (3) as follows: Let $G_s(w_i) = w_iJ_s(w_i)\ \forall w_i$, then Eqn. (3) is equivalent to

$$w^* = \operatorname*{argmin}_{w_i \in (0,w)}\{G_s(w_i) + G_s(w-w_i)\}. \quad (4)$$

Let $L_s(w_i)$ be a continuous and smooth fitting function of $G_s(w_i)$ for $w_i \in [1, w-1]$. Eqn. (3) can then be relaxed by replacing $G_s(w_i)$ with $L_s(w_i)$, which produces the solution at $w^* = w/2$ if $L_s(w_2)$ is convex and $w^* = 1$ if $L_s(w_i)$ is concave. We have observed, over all aggregate functions, the convexity and concavity for $L_s(\cdot)$ when approximating $G_s(\cdot)$. For example, we use 800 game records of a NBA player and plot the function G_s and L_s under various aggregate functions in Fig. 2. In Fig. 2a, G_s and L_s for *min* and *avg* are presented. We can see that the fitting L_s for *min* is convex while that for avg is concave. Similarly, in Fig. 2b, we can see that L_s is concave for count, sum and max. In the worst case scenario, even when $L_s(\cdot)$ is neither convex nor concave, $J_s(w) < \frac{J_s(1)+(w-1)J_s(w-1))}{w}$ still holds due to Theorem 1. Thus we have the visiting-streak bound stated as:

Theorem 2 (Visiting-Streak Bound). *Given a length $w > 1$, let $J_s(w)$ be*

$$J_s(w) = \frac{J_s(1) + (w-1)J_s(w-1)}{w}, \quad (5)$$

then $J_s(w)$ is a visiting-streak bound, i.e., $J_s(w) \ge \max_t\{V_s(t,w)\}$

Proof. By substitute $w_i = 1$ to the right hand side of Theorem 1, we see this theorem holds. □

In Algorithm 1, $J_s(w)$ is computed incrementally. Initially, $J_s(1)$ is set to be the single event of s with highest value. Then, as the subject s is processed, $J_s(w)$ is computed by Theorem 2. In the case when visiting-streak pruning fails, we update $J_s(w)$ to the maximum length-w streak of s to further tighten the bound.

4.1.3 *Unseen-Streak Pruning*

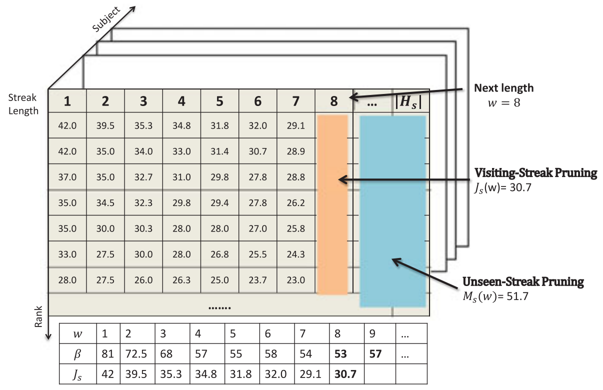
Unseen-streak pruning utilizes $M_s(w)$ to check if it is necessary to evaluate any streak with length $w' \in (w, |\mathbb{H}_s|]$. We observe that $M_s(w)$ can be efficiently estimated from the values of $J_s(w')$, where $w' \le w$. For example, when avg is used as the aggregate function, $J_s(1)$ is obviously an upper bound for $M_s(w)$ because $J_s(1)$ is essentially the maximum event value. However, such an upper bound is very loose. By utilizing the following theorem, we can derive a tighter bound as follows:

Theorem 3 (Unseen-Streak Bound). *Given that $J_s(1), \ldots, J_s(w-1)$ have already been computed, let $M_s(w)$ be*

$$M_s(w) = J_s(w) + \min\left\{\frac{1}{2}J_s(1), \frac{w-1}{w+1}J_s(w-1)\right\}, \quad (6)$$

then $M_s(w)$ is an unseen-streak bound, i.e., $M_s(w) \ge \max\{J_s(w_i)|\, w_i \in [w, |\mathbb{H}_s|]\}$

Proof. First, given any integer $k \ge 1$, we see that $J_s(kw_i) \le J_s(w_i)$ by making use of Theorem 1 in a simple induction.



1	2	3	4	5	6	7	8	...	$\lvert H_s \rvert$
42.0	39.5	35.3	34.8	31.8	32.0	29.1			
42.0	35.0	34.0	33.0	31.4	30.7	28.9			
37.0	35.0	32.7	31.0	29.8	27.8	28.8			
35.0	34.5	32.3	29.8	29.4	27.8	26.2			
35.0	30.0	30.3	28.0	28.0	27.0	25.8			
33.0	27.5	30.0	28.0	26.8	25.5	24.3			
28.0	27.5	26.0	26.3	25.0	23.7	23.0			
....									

w	1	2	3	4	5	6	7	8	9	...
β	81	72.5	68	57	55	58	54	**53**	**57**	...
J_s	42	39.5	35.3	34.8	31.8	32.0	29.1	**30.7**		

Fig. 3. An illustration of streak pruning techniques. Each square slice represents a set of streaks to be computed for a subject, where the column represents streak length and the row represents the rank. The value in each cell is the aggregate result (i.e., $\overline{v}$) for the corresponding streak.

Then, for any integer $x > w$, x can be written as $x = \lfloor \frac{x}{w} \rfloor w + x \bmod w$. Based on the subadditivity of $J_s(\cdot)$, we have

$$\begin{aligned} J_s(x) &\leq \frac{(\lfloor \frac{x}{w} \rfloor w) J_s(\lfloor \frac{x}{w} \rfloor w) + (x \bmod w) J_s(x \bmod w)}{x} \\ &\leq J_s\Big(\Big\lfloor \frac{x}{w} \Big\rfloor w\Big) + \frac{x \bmod w}{x} J_s(x \bmod w), \Big\lfloor \frac{x}{w} \Big\rfloor w \leq x \\ &\leq J_s(w) + \frac{x \bmod w}{x} J_s(x \bmod w), J_s(kt) \leq J_s(t). \end{aligned}$$

On one hand, since $xJ_s(x)$ is a monotone increasing function, it follows $(x \bmod w) J_s(x \bmod w) \leq (w-1) J_s(w-1)$. Moreover, since $x \geq w+1$, we have $(x \bmod w) \frac{J_s(x \bmod w)}{x} \leq (w-1)\frac{J_s(w-1)}{w+1}$. On the other hand, $J_s(x \bmod w) \leq J_s(1)$ and $\frac{x \bmod w}{x} \leq \frac{1}{2}$ for $x > w$, we have $(x \bmod w) \frac{J_s(x \bmod w)}{x} \leq \frac{1}{2} J_s(1)$. Then it naturally leads to Theorem 3. □

To utilize $M_s(w)$, we check if $M_s(w)$ is no greater than any $\beta(w')$ with $w' > w$, i.e., $M_s(w) \leq \min\{\beta(w') | w' \in (w, |\mathbb{H}_s|]\}$. Whenever the condition holds, it is safe to stop further evaluation on subject s. Note that we maintain an interval tree [17] on β to support efficient checking.

Theorem 4. *Each streak returned by Algorithm 1 has a rank no greater than p.*

The proof is quite straightforward according to the descriptions of Algorithm 1, visiting-streak bound and unseen-streak bound. Thus, the details are omitted.

Example 4. We use Fig. 3 to illustrate our pruning techniques on a subject s. Each column in the table represents a streak length and the cells contain the average values among different streaks with different lengths. For example, the cell in the second row and third column refers to the second largest average value among the streaks with length 3. Algorithm 1 essentially accesses the streaks in increasing order of the length. Suppose we are about to estimate the upper bound for streak with length 8. At this point, the values of β and J_s are depicted in the figure. Based on the visiting-streak bound, we estimate $J_s(8) = \frac{42+7*29.1}{8} = 30.7125$. Since $J_s(8) < \beta(8)$, we can safely prune the whole column, as highlighted in the figure. Afterwards, we estimate the upper bound for all the streaks with lengths larger than 8. The value of $M_s(8)$ is then estimated as $M_s(8) = 30.7125 + \min\{21, \frac{7}{9} * 29.1\} = 51.7$. Since all the values of β are greater than $M_s(8)$, it is safe to terminate the streak enumeration.

4.2 k-Sketch Discovery

The second step of our k-Sketch query processing is to, for each subject s, select k ranked-streaks from $\mathcal{N}_s$ to maximize Eqn. (1) (i.e., g). Optimizing g with cardinality constraint (i.e., $|SK|_s = k$) is related to the Partly Interval Set Cover (PISC) problem. The goal in PISC is to select a set of intervals which covers at least a certain percentage of elements. If no rank value is considered in g (i.e., $\alpha = 1$), the solution in [18] for PISC maximizes g in $O(|\mathbb{H}_s|^3)$ time for each subject s. However, when the rank is considered (i.e., $0 < \alpha < 1$), optimizing g becomes an open problem as stated in [19], where current polynomial time solutions remain unknown.

To facilitate scalable k-Sketch discovery, we provide an efficient $(1-1/e)$-approximate algorithm by exploiting the submodularity property[10] of the scoring function. Our idea is that since g is not submodular, it is not easy to directly maximize it. However, we are able to transform g to another submodular function g' s.t. maximizing g' would result in the same optimal solution as maximizing g. We design the function of g' as follows:

$$g'(\mathbb{X}_s) = \eta_1 C'(\mathbb{X}_s) + \eta_2 R'(\mathbb{X}_s), \tag{7}$$

where $C'(\mathbb{X}_s)$ is the number of distinct events covered by $\mathbb{X}_s$, $R'(\mathbb{X}_s) = \Sigma_{X_s \in \mathbb{X}_s} (\mathtt{r} - X_s.r)$, $\eta_1 = \alpha / |\mathbb{H}_s|$ and $\eta_2 = (1-\alpha)/(\mathtt{r}k)$. Given k, s, $\mathtt{r}$ and g, g' is uniquely defined. We have the following theorem which links g' to g:

Theorem 5. *If A^* is the optimal solution under g', then A^* is also the optimal solution under g.*

Proof. First observe that, for any set $A \subseteq \mathbb{N}_s$ of size k, (i.e., $|A| = k$), $g(A) = g'(A)$. This can be validated by substituting A into Eqs. (1) and (7). Then, we prove the theorem by contradiction: Let A^* be the optimal solution under g'. If A^* is not optimal under g, then $\exists B^*$ s.t. $g(B^*) > g(A^*)$. Since $|A^*| = k$, then $g(A^*) = g'(A^*)$. Similarly, since $|B^*| = k$, $g'(B^*) = g(B^*)$. As $g(B^*) > g(A^*)$, it follows $g'(B^*) = g(B^*) > g(A^*) = g'(A^*)$, which contradicts with A^*'s optimality under g'. □

Henceforth, we are able to compute sketches by maximizing g' instead of g. We then prove the *submodularity* on g' as stated below:

Theorem 6. *Given a ranked-streak set $\mathbb{X}_s$, $g'(\mathbb{X}_s)$ is submodular.*

Proof. Note that C' is a cover function and R' is a scalar function, thus C' and R' are both submodular. Since g' a linear combination of C' and R', it is also submodular. □

By utilizing the submodularity of g', we can apply the greedy selection algorithm [20] to efficiently discover the sketches, which guarantees a $(1-1/e)$-approximation ratio. The greedy scheme is presented in Algorithm 2.

10. A function I is submodular if and only if given two set $A \subseteq B$ and an element $x \notin B$, then $I(A \cup \{x\}) - I(A) > I(B \cup \{x\}) - I(B)$).

During each step (Lines 4-7), the algorithm picks the best ranked-streak among the remaining ranked-streaks (i.e., $N_s \setminus SK_s$) to maximizes g'. The algorithm stops at the kth iteration.

Algorithm 2. Greedy Sketch Discovery

```
1: for $s \in S$ do
2:     $\mathbb{N}_s \leftarrow$ ranked-streaks of subject $s$
3:     $SK_s \leftarrow \{\}$
4:     for $t \in [1, k]$ do
5:         $x^* \leftarrow \text{argmax}_{x \in (\mathbb{N}_s \setminus SK_s)} g'(SK_s \cup \{x\})$
6:         $SK_s \leftarrow SK_s \cup \{x^*\}$
7:     end for
8: end for
9: return $SK_s \;\; \forall s \in S$
```

5 Online k-Sketch Maintenance

In the offline scenario, all the events are assumed to be available at the time of k-Sketch query processing. In contrast, in the online scenario, events arrive incrementally. Given an arrival event, our objective is to maintain the k-Sketch for each subject up-to-date. Since events may arrive at a high speed, such a maintenance step has to be done efficiently.

Similar to the offline scenario, we maintain a *WI* to keep track of the top-r streaks for all the possible streak lengths. To handle a newly arrived event $e_s(t)$, a naive solution would first generate all the streaks containing $e_s(t)$, (i.e., $W_s(t, w'), w' \in (1, t]$), and then update *WI* accordingly. Last, Algorithm 2 is invoked to re-compute the sketches. However, there are t associated streaks for each new event $e_s(t)$. Examining all of them is too expensive to support real-time responses. Moreover, Algorithm 2 runs in $O(k|\mathbb{N}_s|)$ time for each affected subject, which imposes further performance challenges.

To achieve instant sketch maintenance, we propose two techniques: *online streak pruning* and *sketch update*. *Online streak pruning* tries to reduce the number of streaks evaluated in generating ranked-streaks. After obtaining the ranked-streaks, we need to update the affected sketches. As we shall see later, given a ranked-streak $N_s(t, w)$, not only the sketch of subject s but also the sketches of other subjects could be affected. Although we provide a solution with a $(1 - 1/e)$ approximation in the offline scenario, to maintain sketches with the same approximation ratio is difficult in the online scenario [21], [22]. Instead, we propose a 1/8-approximate solution which updates a sketch in $O(k)$ time.

Before we present *online streak pruning* and *sketch update*, Algorithm 3 first depicts the overview of our online solution against a new event $e_s(t)$. We iteratively examine streaks which contain $e_s(t)$ (i.e., $W_s(t, w)$ in Line 3). Then we update the sketches which are affected by inserting $W_s(t, w)$ into *WI* (Lines 4-7). Before continuing to examine the next streak length w, we compute the maximum score (i.e., $P_s(w)$) of all streaks which have not been evaluated. If $P_s(w)$ is smaller than all $\beta(w'), w' \in (w, t]$, we can safely stop the process since no further streaks could cause any sketches to change.

Algorithm 3. Online k-Sketch Maintenence

```
Input: $e_s(t) \leftarrow$ arrival event
1: $WI()$// top-r streaks for each length
2: $\beta()$// smallest score in $WI$ for each streak length
3: for $w \in 1, \ldots, t$ do
4:     if $W_s(t, w)$ can be added to $WI(w)$ then
5:         update $\beta(w)$, $J_s(w)$, compute $N_s(t, w)$
6:         SketchUpdate($N_s(t, w)$)
7:     end if
8:     compute $P_s(w)$
9:     break if $P_s(w) \le \max\{\beta(w') | w < w' \le t\}$
10: end for
```

5.1 Online Streak Pruning

Since there are t streaks associated with each new event $e_s(t)$, we wish to avoid enumerating all the possible cases. We achieve the online streak pruning by leveraging the online-streak bound denoted by $P_s(w)$, which is the upper bound value among streaks with lengths greater than w. The value of $P_s(w)$ is stated as in the following theorem:

Theorem 7 (Online-Streak Bound). *Let $W_s(t, 1), \ldots, W_s(t, w)$ be the w streaks computed in Algorithm 3 containing event $e_s(t)$. Let $P_s(w)$ be*

$$P_s(w) = \frac{w}{w+1} V_s(t, w) + min\left\{\frac{t-w}{w+1} J_s(t-w), \frac{t-w}{t} J_s(1)\right\}. \tag{8}$$

Where $J_s(\cdot)$ is the visiting-streak bound. Then $P_s(w)$ is the online-streak bound, i.e., $P_s(w) \ge \max\{V_s(t, x) | x \in (w, t]\}$.

Proof. First, $\forall x \in (w, t]$, we have

$$\begin{aligned} V_s(t, x) &= \frac{wV_s(t, w) + (x - w)V_s(t - w, x - w)}{x} \\ &\le \frac{wV_s(t, w)}{w+1} + \frac{(x - w)V_s(t - w, x - w)}{x}. \end{aligned}$$

Note that $J_s(x - w) \ge V_s(t - w, x - w)$, and $yJ_s(y)$ monotonically increases wrt. y. It follows that $(x - w)V_s(t - w, x - w)/x \le (x - w)J_s(x - w)/x \le (t - w) J_s(t - w)/(w + 1)$. On the other hand, $J_s(1) \ge V_s(\cdot, y)$, for any y. Therefore, $(x - w)V_s(t - w, x - w)/x \le (x - w) J_s(1)/x \le (t - w)J_s(1)/t$. Combining the above deductions, it follows that

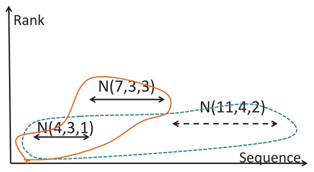
$$\frac{(x - w)V_s(t - w, x - w)}{x} \le min\left\{\frac{t-w}{w+1} J_s(t-w), \frac{t-w}{t} J_s(1)\right\},$$

which leads to Theorem 7. □

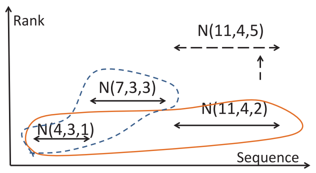
When w is small, $\frac{t-w}{w+1} J_s(t - w)$ is too loose as $\frac{t-w}{w+1}$ is large. However, we can leverage $\frac{t-w}{t} J_s(1)$ to obtain a better bound. As w increases, $\frac{t-w}{w+1} J_s(t - w)$ eventually becomes smaller than $\frac{t-w}{t} J_s(1)$. Thus, we can leverage $\frac{t-w}{w+1} J_s(t - w)$ to perform efficient pruning.

5.2 Sketch Update

Once we obtain a streak $W_s(t, w)$ which causes changes in the $WI(w)$, two kinds of sketch updates may occur. The first update is directly on the sketch of s, which we refer to as



(a) Active update due to new ranked-streak $N(11, 4)$



(b) Passive update due to existing ranked-streak $N(11, 4)$

Fig. 4. The illustration of active updates and passive updates. The solid circle represents the original sketch and the dotted circle represents the updated sketch.

active update. The second update is on the sketches for other subjects. This happens when some of their ranked-streaks become worse due to $W_s(t, w)$. We refer to this case as *passive update*. If these updates are not properly handled, sketches maintained for those subjects are not able to obtain an approximation ratio on their qualities. We demonstrate the two types of updates in the following example.

Example 5. Suppose $k = 2$ and we maintain a two-Sketch for each subject. As shown in Fig. 4a, when the ranked-streak $N(11, 4)$ is generated, the maintained sketch is no longer the best. This is because replacing $N(7, 3)$ would generate a better sketch. This process is referred as the *active update*. In Fig. 4b, $N(11, 4)$ is pushed up due to the arrival of the event about another subject; as a result, the quality of the sketch drops. We name this process as the *passive update*. If passive update is not handled, the rank of $N(11, 4)$ may continue to be pushing up and may eventually be greater than p, making the entire sketch invalid. Nevertheless, it is evident that when $N(11, 4)$ degrades, replacing it with $N(7, 3)$ would result in a sketch with a better quality.

A naive approach to handle these updates is to run Algorithm 2 for each affected subject. This maintains a $(1 - 1/e)$-approximation ratio but incurs a high computational cost. In order to support efficient updates, we make a trade-off between the quality of the sketch and the update efficiency by providing a $1/8$-approximate solution with only $O(k)$ ranked-streaks being accessed for each affected subject.

Algorithm 4. SketchUpdate

```
Input: $N_s(t, w)$
1:  Active update for the subject $s$
2:  $S_1$: $k$ ranked-streaks with best cover
3:  $S_2$: $k$ ranked-streaks with best ranks
4:  $N^* \leftarrow argmax_{N \in S_1} C(S_1 \cup N_s(t, w) \setminus N)$
5:  if $C(S_1) < C(S_1 \cup N_s(t, w) \setminus N^*)$ then
6:      $S_1 \leftarrow S_1 \cup N_s(t, w) \setminus N^*$
7:  end if
8:  $S_2 \leftarrow$ new $k$ ranked-streaks with best ranks
9:  $S \leftarrow greedy(S_1 \cup S_2)$
10:
11: Passive update for an affecting subject $s'$
12: $S_2 \leftarrow$ new $k$ ranked-streaks with best ranks for $s'$
13: $S \leftarrow greedy(S_1 \cup S_2)$
```

In particular, we maintain two size-k sets S_1 and S_2. S_1 maintains the k best ranked-streaks which collectively cover most events whereas S_2 maintains k ranked-streaks with best ranks. When performing an active update for a streak $N_s(t, w)$, we check if $N_s(t, w)$ could replace any ranked-streak in S_1 to generate a better cover. Meanwhile, we select the new k best ranked-streaks into S_2. After S_1 and S_2 are updated, we perform the greedy selection from $S_1 \cup S_2$. During a passive update, S_1 is not affected. We simply update S_2 to be the new k best ranked-streaks. Afterwards, the new sketch is obtained by performing a greedy selection from $S_1 \cup S_2$. Algorithm 4 presents both the active and passive updates.

We state the quality of our sketch update strategy in the following theorem:

Theorem 8 (Approximation Ratio for Sketch Update). *Each sketch maintained by Algorithms 3 achieves an at least $1/8$-approximation to the optimal solution.*

Proof. First, we observe that S_2 always keeps the ranked-streaks with optimal ranks. Second, we note that S_1 maintains the streaks with $1/4$-approximation as shown in [23].

Let OPT_C be the optimal k streaks that best covers s's history; Let $C()$ be the number of events a set of streaks cover. Similarly, let OPT_R be the optimal k streaks with highest ranks; Let $R()$ be the summation of ranks of all members in a ranked-streak set. Let S_s^* be the optimal sketch of subject s. Intuitively, we have the following:

$$g'(S_s^*) \leq \eta_1 C(OPT_C) + \eta_2 R(OPT_R).$$

Since $C(S_1) \geq 1/4C(OPT_C)$ and $R(S_2) = R(OPT_R)$, we have the following:

$$\begin{aligned}\eta_1 C(S_1) + \eta_2 R(S_2) &\geq 1/4 * \eta_1 C(OPT_C) + \eta_2 R(OPT_R) \\ &\geq 1/4 g'(S_s^*),\end{aligned}$$

which implies $\max\{\eta_1 C(S_1), \eta_2 R(S_2)\} \geq 1/8 g(S_s^*)$. As $g'(S_1) \geq \eta_1 C(S_1)$ and $g'(S_2) > \eta_2 R(S_2)$, it leads to

$$\max\{g'(S_1), g'(S_2)\} > 1/8 g'(S_s^*).$$

Let SK_s be one of the sketch maintained by Algorithms 4, since the greedy algorithm is run on $S_1 \cup S_2$, $g'(SK_s) \geq max(g'(S_1), g'(S_2)) \geq 1/8 g'(S_s^*)$. As a result, our algorithm always ensures at least $1/8$-approximation for each sketch. □

6 Experiments

In this section, we study our solutions for k-Sketch query processing in both offline and online scenarios using the following four real datasets. The statistics of these datasets are summarized in Table 4.

NBA[11] contains the game records for each NBA player from year 1985 to 2013. Among all the records, we pick 1,000 players with at least 200 game records. In total, we obtain a dataset with 569 K events.

POWER [24] contains the electricity usage for 370 households between Dec. 2006 and Nov. 2010. Each household is treated as a subject with the daily power usage as an event. In total, there are 1.4 M events.

11. http://www.nba.com

TABLE 4
Statistics of Datasets Used in the Experiments

DataSet	Total Events	Total Subjects	Longest Sequence
NBA	569,253	1,015	1,476
POWER	1,480,000	370	4,000
PEMS	5,798,918	963	6,149
STOCK	2,326,632	318	10,420

PEMS [25] contains the occupancy rate of freeway in San Francisco bay area from Jan. 2008 to Mar. 2009. Each freeway is a subject with the daily occupancy rate as an event. The dataset contains 963 freeways with 5.7 M events.

STOCK contains the hourly price tick for 318 stocks from Mar. 2013 to Feb. 2015. The dataset is crawled from Yahoo! Finance[12] and contains 2.3 M events.

In our efficiency study, we evaluate three parameters: (1) $\mathtt{r} \in [20, 200]$, which refers to the threshold of the ranked-streaks, (2) $k \in [20, 100]$, which refers to the size of a sketch and (3) $h \in [20, 100]$, which refers to the percentage of historical events for scalability test, i.e., $|\mathbb{H}|h\%$ events are used in the experiments. We do not evaluate the performance wrt. α since α does not impact the running time. We use $\mathtt{r} = 200$, $h = 100$, and $k = 20$ as the default values.

All the experiments are conducted on a desktop machine equipped with an Intel i7 Dual-Core 3.0 GHz CPU, 8 GB memory and 160 GB hard drive. All algorithms are implemented in Java 7.

6.1 Offline k-Sketch Query Processing

Our offline k-Sketch query processing algorithms consist of two functional components, *Ranked-streak Generation* and *Sketch Discovery*. In the ranked-streak generation, we report the performance with varying $\mathtt{r}$ and h. In the sketch discovery, we report the performance with varying k.

6.1.1 Ranked-Streak Generation Algorithms

To evaluate the performance, we design the following four methods for comparison:

Brute Force (BF). BF exhaustively computes and compares for each subject all the possible streak lengths.

Visiting Streak Pruning (V-SP). V-SP only adopts the *visiting-streak* bound for pruning.

Unseen Streak Pruning (U-SP). U-SP only adopts the *unseen-streak* bound for pruning.

Unseen+Visiting Streak Pruning (UV-SP). UV-SP adopts both *unseen-streak* and *visiting-streak* bounds for pruning.

6.1.2 Ranked-Streaks Generation with Varying p

The running time of the four algorithms in ranked-streak generation wrt. $\mathtt{r}$ is shown in Fig. 5. It is evident that when $\mathtt{r}$ increases, more ranked-streaks are qualified and thus all four algorithms require more computation time. The effect of the two proposed streak-based pruning techniques can also be observed from the figure. The insight is that the unseen-streak pruning plays a more important role in reducing the running time. Furthermore, when both pruning techniques are used, our method (UV-SP) achieves at least two orders of magnitude of performance improvement.

12. https://finance.yahoo.com/

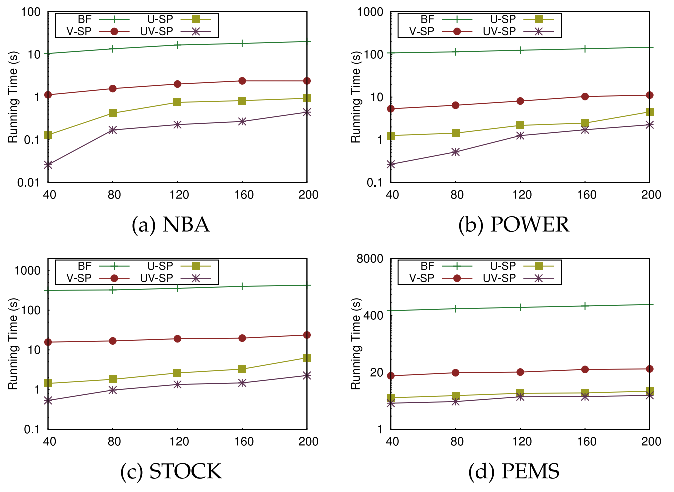


Fig. 5. Ranked-streak generation in the offline scenario with varying $\mathtt{r}$.

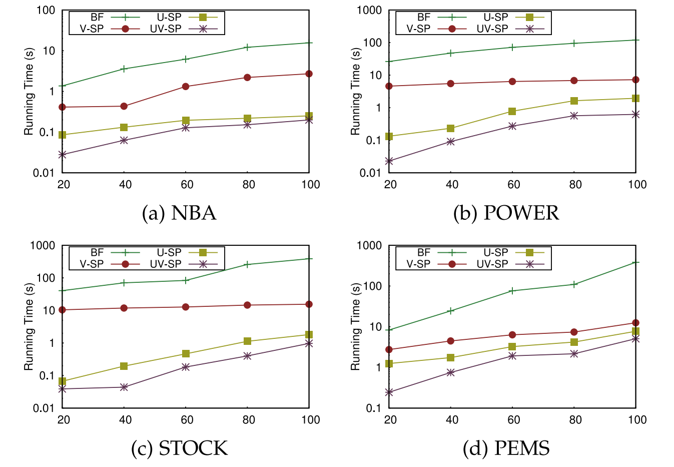


Fig. 6. Ranked-streak generation in the offline scenario with varying h.

6.1.3 Ranked-Streak Generation with Varying h

We then study the performance of the four algorithms wrt. the number of events and report the results in Fig. 6. As the figure shows, when h increases, the running times for all four algorithms increase. This is because more streaks need to be evaluated. Again, pruning-based methods are much more efficient than the baseline method. When both pruning methods are adopted, our method (UV-SP) obtains hundreds of times faster than the baseline method.

6.1.4 Sketch Discovery with Varying k

After ranked-streaks are generated, we greedily find the k-sketch for each subject. Here, we study the effect of k on the performance of the greedy algorithm. The results on the four datasets are presented in Table 5. The table indicates that the running time of the greedy algorithm increases proportionally to k. This is because the complexity of the greedy algorithm is $O(k\Sigma_s|\mathbb{N}_s|)$. Since PEMS is the largest dataset with highest $\Sigma_s|\mathbb{N}_s|$, the greedy algorithm performs worst on PEMS. We also observe that the greedy algorithm takes near 500 seconds on PEMS, which implies that the performance of exact solutions with cubic complexity is not acceptable. This confirms the necessity of adopting the approximate algorithm.

TABLE 5
Sketch Discovery with Varying k in (ms)

k	20	40	60	80	100
NBA	9,097	13,345	17,500	21,597	30,769
POWER	36,297	53,513	69,300	86,603	122,856
STOCK	63,679	93,415	122,500	151,179	215,386
PEMS	138,224	206,820	283,190	353,766	491,000

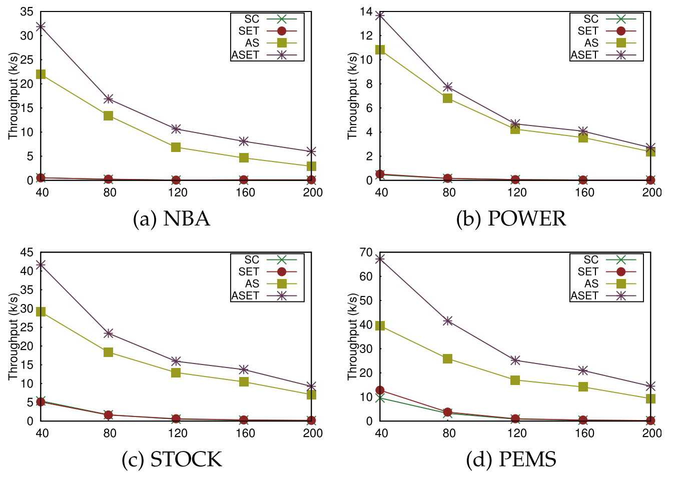


Fig. 7. Throughput in online scenario with varying r.

6.2 Online Sketch Maintenance

In the online scenario, we evaluate the following four algorithms in our performance study:

Sketch Computing (SC). SC examines all streaks generated from a fresh event. Then Algorithm 2 is invoked for each affected subject. To improve efficiency, the updates are processed in batches, i.e., multiple updates on the same subject will be batched and processed by calling Algorithm 2 once.

Sketch with Early Termination (SET). SET adopts *online-streak* bound in Theorem 7 for early termination.

Approx. Sketch (AS). AS is similar to *SC* except that it only computes the approximate sketches.

Approx. Sketch with Early Termination (ASET). ASET computes the approximate sketches with early termination, as shown in Algorithm 3.

In the online setting, we are more interested in evaluating the throughput of algorithms. We report the performance wrt. p, k and h.

6.2.1 Query Throughput with Varying r

We increase r from 10 to 200 and the throughput results are shown in Fig. 7. The figure demonstrates similar patterns for all four datasets. As r increases, the throughput of the four algorithms drops. This is because as r increases, the time required to maintain the top-r ranked-streaks as well as to update the sketch increases. However, algorithms adopting *online-streak bound* have higher throughput than their counterparts. This is because with early termination, fewer streaks are generated. We can also see that *SC* and *ASC* run very slowly in the online setting. This is because they need to invoke Algorithm 2 upon every update. This confirms the necessity of our approximate method as *ASET* achieves up to 500x speedup as compared to *SC*.

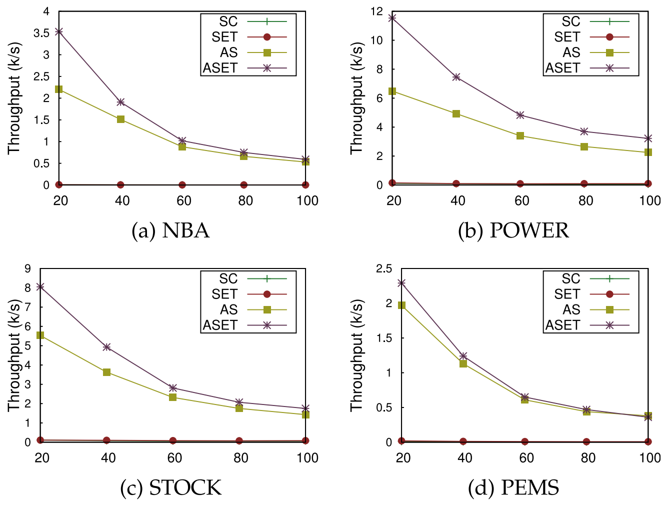


Fig. 8. Throughput in the online scenario with varying k.

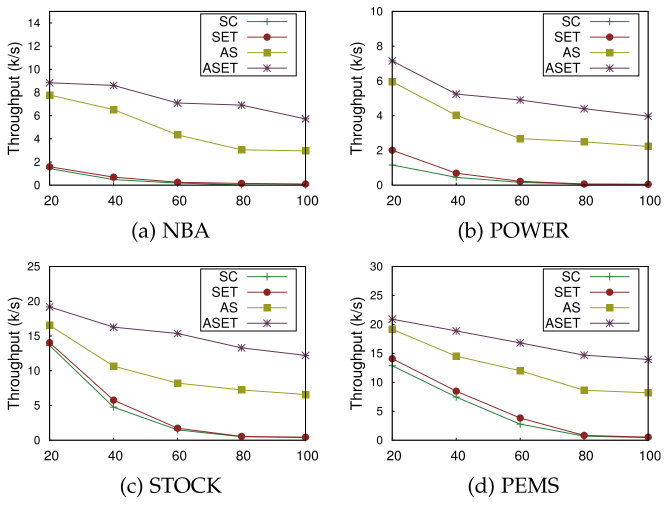


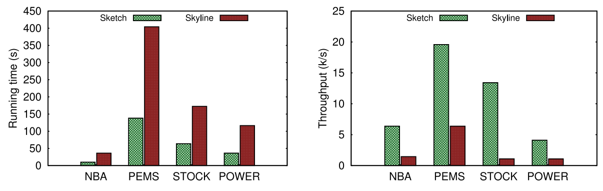
Fig. 9. Throughput in the online scenario with varying h.

6.2.2 Query Throughput with Varying k

We then evaluate how the throughput varies wrt. k. The results are presented in Fig. 8. The figure tells similar patterns as Fig. 7. First, as k increases, the throughput of all four algorithms decreases. This is because as k becomes larger, more operations are needed for maintaining the sketch. Second, the throughput of *SC* and *SET* are an order of magnitude smaller than *AS* and *ASET*. This is because *SC* and *SET* repetitively call Algorithm 2 which heavily depends on k. We observe that in some datasets (e.g., Fig. 8a) there is 100x boost for *ASET* as compared to *SC*.

6.2.3 Query Throughput with Varying h

Finally, we study the effect of h in affecting the throughput. We change h from 20 to 100, and the results are represented in Fig. 9. As shown in the figure, when h increases, the throughput of the four algorithms drops steadily. This is because as h increases, $|\mathbb{H}_s|$ for each subject increases. Therefore in Algorithm 3, more time is needed to process each streak. We notice that *ASET* has a flatter slope than *AS*; this is benefit from the prunings of *online-streak* bound.



(a) Offline Scenario (b) Online Scenario

Fig. 10. Efficiency comparison with prominent streaks.

TABLE 6
Summaries of "Kobe Bryant" on Point-per-Game-Average (PPGA) Obtained by the Four Methods

Method	Career Summaries Generated
SY_m (SY_r)	1. **2006/04/16**, Kobe obtained **37.05** PPGA in **57** straight games (ranked **10** in NBA history)!
	2. **2006/04/16**, Kobe obtained **35.09** PPGA in **85** straight games (ranked **111** in NBA history)!
	3. **2006/04/19**, Kobe obtained **36.91** PPGA in **61** straight games (ranked **12** in NBA history)!
	4. **2006/11/03**, Kobe obtained **35.85** PPGA in **75** straight games (ranked **63** in NBA history)!
	5. **2006/11/08**, Kobe obtained **35.29** PPGA in **78** straight games (ranked **85** in NBA history)!
SK_a	1. **2003/03/11**, Kobe obtained **38.05** PPGA in **20** straight games, ranked **25** in NBA history!
	2. **2006/02/08**, Kobe obtained **38.67** PPGA in **28** straight games, ranked **1** in NBA history!
	3. **2006/02/26**, Kobe obtained **38.10** PPGA in **30** straight games, ranked **1** in NBA history!
	4. **2006/04/19**, Kobe obtained **37.17** PPGA in **56** straight games, ranked **5** in NBA history!
	5. **2007/10/30**, Kobe obtained **32.27** PPGA in **212** straight games, ranked **194** in NBA history!
SK	1. **2003/02/21**, Kobe obtained **43.20** PPGA in **10** straight games, ranked **3** in NBA history!
	2. **2006/01/22**, Kobe obtained **81.00** PPGA in **1** straight games, ranked **1** in NBA history!
	3. **2006/02/11**, Kobe obtained **38.24** PPGA in **29** straight games, ranked **1** in NBA history!
	4. **2007/03/30**, Kobe obtained **46.12** PPGA in **8** straight games, ranked **1** in NBA history!
	5. **2008/02/01**, Kobe obtained **32.24** PPGA in **216** straight games, ranked **195** in NBA history!

SY_r corresponds to the same SY_m streak with augmented rank information.

6.3 Comparison with Other Techniques

We also compare the efficiency and effectiveness of our k-Sketch query with the state-of-the-art Prominent Streaks query [4] (denoted by the skyline method) in providing newsworthy summaries.

To study the efficiency, we implement the skyline algorithms as described in [4] for both online and offline scenarios. The results under all four datasets are presented in Fig. 10. The figure demonstrates the superiority of our schemes in both scenarios. Specifically, our offline scheme saves 63 to 75 percent processing time and our online scheme achieves 2 to 10 time throughput speedups. These results further indicate the efficiency of our schemes.

To study the effectiveness, we conduct a user study over Amazon Mechanical Turk[13] to evaluate the attractiveness of the summaries generated by different methods. For our method, we set $\alpha = 0.5$ to pay equal attention to the strikingness and the coverage. For the skyline method, due to the overwhelming skylines generated for each subject, we propose two augmented methods to pick k of them. In total, we compare the following four algorithms:

(1) SK: selects the k-sketch for each player generated by the offline k-Sketch query.
(2) SK_a: selects the k-sketch for each player generated by the online k-Sketch query.
(3) SY_m: randomly selects k streaks for each player from the bunch of skylines generated by [4].
(4) SY_r: attaches ranks[14] to the streaks in SY_m.

We then apply each method on the NBA dataset and set $k = 5$ to generate 5 streaks for each player. Each streak is then translated into a news theme in the following format:

2003/04/14: Jordan obtained 30.3 PPGA[15] in 989 straight games, ranked 1st in NBA history!!

We design each job in AMT to contain five questions and each question presents the four summaries of a player generated by the four methods. A sample question regarding "Kobe Bryant" is listed in Table 6. Due to space limitation, we present SY_m and SY_r in one row, since they essentially report the same streak, except that SY_r provides additional rank information. We then ask the respondents to endorse each summary based on the level of attractiveness. We receive responses from 100 participants who have knowledge in NBA.[16] Then, for each algorithm, we count the frequency of it being endorsed as the best and report the percentage results in the pie chart in Fig. 11.

The chart clearly shows that SK is the most effective method as it takes 51 percent of the endorsements from the respondents. Overall, Sketch based methods (i.e., SK and SK_a) receive 75 percent endorsements, which win the skyline methods (i.e., SY_m and SY_r) by three times. The chart also shows that, when applied with the rank information (i.e., SY_r), the number of endorsements increases dramatically, more than two times of the original number of endorsements (i.e., SY_m). This also implies the effectiveness of our ranked-streaks. We can also observe the quality differences of the four methods in Table 6. As the table shows, SY_m and SY_r output streaks concentrating on a shorter period (i.e., 2006) as compared with the output of SK_a and SK (i.e., 2003-2008). This is because Kobe unprecedentedly

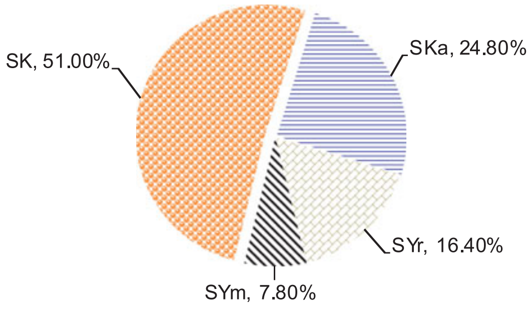


Fig. 11. Percentage of news endorsed as the most attractive.

13. https://requester.mturk.com

14. Rank is generated by our offline method.
15. PPGA: Point-per-game-average.
16. In AMT, we are able to request respondents with certain qualifications, i.e., knowledgeable in NBA.

TABLE 7
Five-Sketches for "Dominique-Wilkins" from NBA Dataset wrt α

α	Sketches			
	Date	Streak Length	Avg (points)	Rank
0.1	1986-04-01	2	47.00	14
	1986-04-10	1	57.00	11
	1987-02-10	2	50.00	8
	1988-03-01	2	48.50	11
	1988-03-01	11	40.54	16
0.5	1986-04-01	2	47.00	14
	1987-02-10	2	50.00	8
	1988-03-01	2	48.50	11
	1988-03-11	14	39.35	18
	1991-12-10	3	44.00	38
0.9	1986-11-02	55	32.98	147
	1987-03-26	26	32.46	145
	1987-02-10	14	32.21	117
	1988-04-19	61	33.19	139
	1993-03-29	34	32.88	145

scored 81 points on 2006/01/22, thus most skylines are associated with that event. Moreover, the streaks selected by SY_m and SY_r are not ranked well as compared with the results of SK_a and SK. The reason is that skyline based methods only consider the local prominence (i.e., non-dominated in one's career) but k-Sketch considers the global prominence (i.e., rank in history).

6.4 Case Study on Real Data

We further study the k-Sketch query in the NBA dataset. We compute the five-sketch for the player "Dominique-Wilkins" using different α and list the ranked-streaks in Table 7. As the table shows, when α is small (i.e., 0.1), the streaks selected tend to have higher ranks. On the other hand, when α is large (i.e., 0.9), the streaks have larger coverage.

We observe several interesting facts from the sketches. First, the streaks with highest coverage (i.e., $\alpha = 0.9$) concentrate in the period 1986-1993, while the highest ranked-streaks (i.e., $\alpha = 0.1$) locate in the period 1986-1988. Looking up the ground truth,[17] we find that "Dominique"'s prime career is in 1985-1993 and he was selected into the All-Star team every year in 1986-1988, which is consistent with our discoveries. Second, our k-Sketch query also discovers a length-1 streak of 57 points (in 1986-04-10, ranked 11th in history), which is in fact the career-highest points scored by "Dominique".[18] Last, we notice that "Dominique" has a length-2 streak with an average score of 50 points, which ranks 8th in history. Although this streak ranks better than the length-1 streak with 57 points, interestingly, it has not been reported in any news. This indicates that our k-Sketch query is able to discover interesting facts where human experts may miss.

7 Supporting Other Aggregate Functions

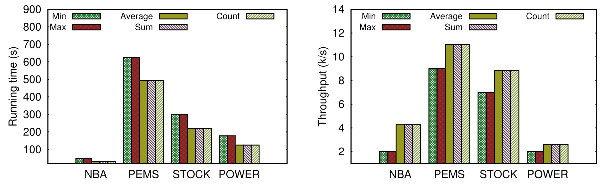
First, we shall see that supporting *sum* is equivalent to supporting *avg*. A ranked-streak which has a rank under *avg* will have the same rank under *sum* as the ranking is derived by comparing all candidates with the same length. Second, supporting *count* is equivalent to supporting *sum*. By assigning each event with a value of either 1 or 0, we can apply the same pruning bounds for *sum* to support *count*. Third, supporting *max* is equivalent to supporting *min*. This is because when *max* is used as the aggregate function, we are more interested to find streaks which have smaller aggregation values. For example, "XXX stock has a maximum of \$0.2 price in consecutive 10 days, which is the lowest ever". Then finding the sketches according to *max* can be derived from *min* directly by negating the event values. Therefore, we only provide bounds for *sum* and *min*, which are shown as in Table 8.

We present the performance variations of our k-Sketch query under different aggregate functions in Fig. 12. We can see from the figures that when adopting *min* (*max*) the performance in both online and offline scenarios drops (20 to 30 percent). This indicates that the pruning bounds in *min* (*max*) is weaker than *avg* (*sum*, *count*).

17. http://www.nba.com/history/players/wilkins_stats.html
18. http://articles.latimes.com/1986-12-11/sports/sp-2180_1_22-point-deficit

TABLE 8
Bounds for Other Aggregate Functions

Aggregate Function	Subadditivity
sum	$J_s(w) \leq J_s(w_1) + J_s(w - w_1)$
min	$J_s(w) \leq max(J_s(w_1), J_s(w - w_1))$
Aggregate Function	**Visting-Streak Bound**
sum	$J_s(w) = J_s(w-1) + J_s(1)$
min	$J_s(w) = J_s(w/2)$
Aggregate Function	**Unseen-Streak Bound**
sum	$M_s(w) = V_s(t, w) + J_s(t - w)$
min	$M_s(w) = max\{V_s(t, w), J(1)\}$
Aggregate Function	**Online-Streak Bound**
sum	$M_s(w) = V_s(t, w) + J_s(t - w)$
min	$M_s(w) = max\{V_s(t, w), J(1)\}$



(a) Offline Scenario (b) Online Scenario

Fig. 12. Performance under different aggregate functions.

8 Conclusion and Future Works

In this paper, we look at the problem of automatically summarizing a subject's history with newsworthy stories. We group consecutive events into streaks and propose a novel idea of *ranked-streak* to represent the strikingness. We then formulate the *k-Sketch* query which aims to best summarize a subject's history using k ranked-streaks. We study the k-Sketch query processing in both offline and online scenarios, and propose efficient solutions to cope each scenario. In particular, we design novel streak-level pruning techniques and a $(1 - 1/e)$-approximate algorithm to achieve efficient processing in offline. Then we design a $1/8$-approximate algorithm for the online sketch maintenance. Our comprehensive experiments demonstrate the efficiency of our solutions and a human study confirms the effectiveness of the k-Sketch query.

Our work opens a wide area of research in computational journalism. In our next step, we aim to extend the event sources to support non-schema data such as tweets in social networks. We also plan to investigate subject association models to automatically generate news themes involving multiple subjects. We further wish to leverage big data technology to support fast-growing event data in journalism.

ACKNOWLEDGMENTS

The authors would like to thank all of the anonymous reviewers for their responsible and detailed feedback. This work is supported in part by the National Nature Science Foundation of China under grants No. 61602087 and No. 61632007.

REFERENCES

[1] A. Sultana, N. Hassan, C. Li, J. Yang, and C. Yu, "Incremental discovery of prominent situational facts," in *Proc. IEEE 30th Int. Conf. Data Eng.*, 2014, pp. 112–123.
[2] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu, "On one of the few objects," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 1487–1495.
[3] N. Hassan, et al., "Data in, fact out: Automated monitoring of facts by FactWatcher," *Proc. VLDB Endowment*, vol. 7, no. 13, pp. 1557–1560, 2014.
[4] G. Zhang, X. Jiang, P. Luo, M. Wang, and C. Li, "Discovering general prominent streaks in sequence data," *ACM Trans. Knowl. Discovery Data*, vol. 8, no. 2, 2014, Art. no. 9.
[5] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of frequent episodes in event sequences," *Data Mining Knowl. Discovery*, vol. 1, no. 3, pp. 259–289, 1997.
[6] W. Zhou, H. Liu, and H. Cheng, "Mining closed episodes from event sequences efficiently," in *Advances in Knowledge Discovery and Data Mining*. Berlin, Germany: Springer, 2010, pp. 310–318.
[7] N. Tatti and B. Cule, "Mining closed strict episodes," *Data Mining Knowl. Discovery*, vol. 25, no. 1, pp. 34–66, 2012.
[8] S. Laxman, P. Sastry, and K. Unnikrishnan, "A fast algorithm for finding frequent episodes in event streams," in *Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2007, pp. 410–419.
[9] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong, "Diversifying search results," in *Proc. 2nd ACM Int. Conf. Web Search Data Mining*, 2009, pp. 5–14.
[10] A. Borodin, H. C. Lee, and Y. Ye, "Max-sum diversification, monotone submodular functions and dynamic updates," in *Proc. 31st Symp. Principles Database Syst.*, 2012, pp. 155–166.
[11] M. Drosou and E. Pitoura, "Diverse set selection over dynamic data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 5, pp. 1102–1116, May 2014.
[12] L. Chen and G. Cong, "Diversity-aware top-k publish/subscribe for text stream," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 347–362.
[13] J. Allan, R. Papka, and V. Lavrenko, "On-line new event detection and tracking," in *Proc. 21st Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 1998, pp. 37–45.
[14] T. Brants, F. Chen, and A. Farahat, "A system for new event detection," in *Proc. 26th Annu. Int. ACM SIGIR Conf. Res. Develop. Informaion Retrieval*, 2003, pp. 330–337.
[15] X. Li, H. Cai, Z. Huang, Y. Yang, and X. Zhou, "Social event identification and ranking on flickr," *World Wide Web*, vol. 18, no. 5, pp. 1219–1245, 2015.
[16] J. B. Vuurens, A. P. de Vries, R. Blanco, and P. Mika, "Online news tracking for ad-hoc information needs," in *Proc. Int. Conf. Theory Inf. Retrieval*, 2015, pp. 221–230.
[17] M. D. Berg, O. Cheong, M. V. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Santa Clara, CA, USA: Springer-Verlag, 2008.
[18] L. Golab, H. Karloff, F. Korn, A. Saha, and D. Srivastava, "Sequential dependencies," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 574–585, 2009.
[19] K. Edwards, S. Griffiths, and W. S. Kennedy, "Partial interval set cover–trade-offs between scalability and optimality," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Berlin, Germany: Springer, 2013, pp. 110–125.
[20] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—I," *Math. Program.*, vol. 14, no. 1, pp. 265–294, 1978.
[21] N. Alon, B. Awerbuch, and Y. Azar, "The online set cover problem," in *Proc. 35th Annu. ACM Symp. Theory Comput.*, 2003, pp. 100–105.
[22] B. Awerbuch, Y. Azar, A. Fiat, and T. Leighton, "Making commitments in the face of uncertainty: How to pick a winner almost every time," in *Proc. 28th Annu. ACM Symp. Theory Comput.*, 1996, pp. 519–530.
[23] B. Saha and L. Getoor, "On maximum coverage in the streaming model & application to multi-topic blog-watch," in *Proc. SIAM Int. Conf. Data Mining*, 2009, pp. 697–708.
[24] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml
[25] T. Choe, A. Skabardonis, and P. Varaiya, "Freeway performance measurement system: Operational analysis tool," *Transp. Res. Rec.: J. Transp. Res. Board*, vol. 1811, pp. 67–75, 2002.

Qi Fan received the bachelor of computing (first class Hons.) degree from the School of Computing, National University of Singapore, in 2012. He is working toward the PhD degree at the National University of Singapore's Graduate School of Integrative Science and Engineering (NGS). He is supervised by Prof. Kian-Lee Tan. His research interests include graph analytics, pattern mining, and computational journalism.

Yuchen Li received the double (both degrees with first class Hons.) degrees in applied math and computer science from the National University of Singapore, in 2012 and the PhD degree from the National University of Singapore (NUS), in 2017. He is a research fellow in the School of Computing, NUS. He worked as a research assistant in the School of Computing, NUS, from 2012 to 2013 and from 2016 to 2017. His research interests includes social network data management, search, and analysis.

Dongxiang Zhang received the BSc degree from Fudan University, China, in 2006 and the PhD degree from the National University of Singapore, in 2012. He is a professor with the University of Electronic Science and Technology of China. He worked as a research fellow at the NeXT Research Center in Singapore from 2012 to 2014, and he was promoted to senior research fellow in 2015. His research interests include spatial information retrieval, moving object queries, and multimedia indexing.

Kian-Lee Tan received the PhD degree in computer science from the National University of Singapore (NUS), in 1994. He is a professor of computer science in the School of Computing, NUS. His current research interests include query processing and optimization in multiprocessor and distributed systems, database performance, data analytics, and database security. He has published more than 300 research articles in international journals and conference proceedings, and co-authored several books/monographs. He received the NUS Outstanding University Researchers Award in 1998, and the NUS Graduate School (NGS) Excellent Mentor Award in 2011. He was a co-recipient of Singapore's President Science Award in 2011. He is also a 2013 IEEE Technical Achievement Award recipient. He is a member of the VLDB Endowment Board and PVLDB Advisory Committee. He is an associate editor of the *ACM Transactions on Database Systems* and the *WWW Journal*. He has also served on the editorial boards of the *Very Large Data Base Journal* (associate editor: 2007-2009; editor-in-chief: 2009-2015) and the *IEEE Transactions on Knowledge and Data Engineering* (2009-2013). He is a member of the ACM the IEEE, and the IEEE Computer Society.