Prominent Streak Discovery in Sequence Data

Xiao Jiang[*]
Shanghai Jiao Tong University
showbufire@gmail.com

Chengkai Li[† ‡]
University of Texas at Arlington
cli@uta.edu

Ping Luo
HP Labs China
ping.luo@hp.com

Min Wang
HP Labs China
min.wang6@hp.com

Yong Yu
Shanghai Jiao Tong University
yyu@cs.sjtu.edu.cn

ABSTRACT

This paper studies the problem of prominent streak discovery in sequence data. Given a sequence of values, a prominent streak is a long consecutive subsequence consisting of only large (small) values. For finding prominent streaks, we make the observation that prominent streaks are skyline points in two dimensions– streak interval length and minimum value in the interval. Our solution thus hinges upon the idea to separate the two steps in prominent streak discovery– candidate streak generation and skyline operation over candidate streaks. For candidate generation, we propose the concept of local prominent streak (LPS). We prove that prominent streaks are a subset of LPSs and the number of LPSs is less than the length of a data sequence, in comparison with the quadratic number of candidates produced by a brute-force baseline method. We develop efficient algorithms based on the concept of LPS. The non-linear LPS-based method (NLPS) considers a superset of LPSs as candidates, and the linear LPS-based method (LLPS) further guarantees to consider only LPSs. The results of experiments using multiple real datasets verified the effectiveness of the proposed methods and showed orders of magnitude performance improvement against the baseline method.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*data mining*

General Terms

Algorithms, Performance

Keywords

sequence database, time-series database, skyline query

[*]Work performed while the author was visiting HP Labs China.

[†]Work partially performed while the author was visiting HP Labs China.

[‡]This material is based upon work partially supported by NSF Grant IIS-1018865. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of NSF.

KDD'11, August 21–24, 2011, San Diego, California, USA.

1. INTRODUCTION

This paper defines the problem of *prominent streak discovery* in sequence data and presents efficient algorithms for finding prominent streaks. A piece of sequence data is a series of values or events. This includes time-series data, in which the data values or events are often measured at equal time intervals. Sequence and time-series data is produced and accumulated in a rich variety of applications. Examples include stock quotes, sports statistics, temperature measurement, Web usage logs, network traffic logs, Web clickstream, and customer transaction sequence.

Given a sequence of values, a *prominent streak* is a long consecutive subsequence consisting of only large (small) values. Examples of such prominent streaks include consecutive days of high temperature, consecutive trading days of large stock price oscillation, consecutive games of outstanding performance in professional sports, consecutive hours of high volume of TCP traffic, consecutive weeks of high flu activity, and so on. It is insightful to investigate prominent streaks since they intuitively and succinctly capture extraordinary subsequences of data.

Prominent streak discovery can be particularly useful in helping journalists to identify newsworthy stories when data sequences evolve, investigators to find suspicious phenomena, and news anchors and sports commentators to bring out attention-seizing factual statements. Therefore it will be a key enabling technique for *computational journalism* [7]. In fact, we witness the mentioning of prominent steaks in many real-world news articles:

- *"This month the Chinese capital has experienced 10 days with a maximum temperature in around 35 degrees Celsius – the most for the month of July in a decade."* (http://www.chinadaily.com.cn/china/2010-07/27/content_11055675.htm)
- *"The Nikkei 225 closed below 10000 for the 12th consecutive week, the longest such streak since June 2009."* (http://www.bloomberg.com/news/2010-08-06/japanese-stocks-fall-for-second-day-this-week-on-u-s-jobless-claims-yen.html)
- *"He (LeBron James) scored 35 or more points in nine consecutive games and joined Michael Jordan and Kobe Bryant as the only players since 1970 to accomplish the feat."* (http://www.nba.com/cavaliers/news/lbj_mvp_candidate_060419.html)
- *"Only player in NBA history to average at least 20 points, 10 rebounds and 5 assists per game for 6 consecutive seasons. (Kevin Garnett)"* (http://en.wikipedia.org/wiki/Kevin_Garnett)

The examples indicate that general prominent streaks can have a variety of constraints. A streak can be on multiple dimensions (e.g., ⟨point, rebound, assist⟩), its significance can be with regard to a certain period (e.g., "since June 2009") or a certain comparison group (e.g., "the month of July"), and we may be interested in not only the most prominent streaks but also the top-k most prominent ones (e.g., "X joined Y and Z as the only players").

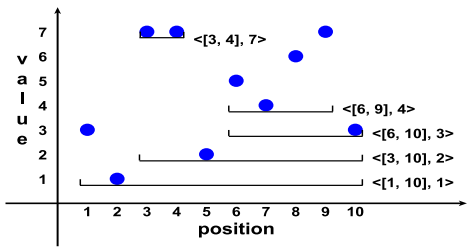


Figure 1: A Data Sequence and its Prominent Streaks.

Given its real-world usefulness and variety, the research on prominent streaks in sequence data opens a spectrum of challenging problems. In this initial effort, we undertake the problem of discovering the simplest kind of prominent streaks, i.e., those without the above constraints. We formally define the concept of prominent streak and our problem as follows.

1.1 Problem Definition

Definition 1 (Streak and Prominent Streak): Given an n-element sequence P=$(p_1, \cdots, p_n)$, a *streak* is an interval-value pair $\langle [l, r], v\rangle$, where $1 \leq l \leq r \leq n$ and v=$\min_{l \leq i \leq r} p_i$.

Consider two streaks s_1=$\langle [l_1, r_1], v_1\rangle$ and s_2=$\langle [l_2, r_2], v_2\rangle$. We say s_1 *dominates* s_2, denoted by $s_1 \succ s_2$ or $s_2 \prec s_1$, if $r_1 - l_1 \geq r_2 - l_2$ and $v_1 > v_2$, or $r_1 - l_1 > r_2 - l_2$ and $v_1 \geq v_2$.

With regard to P=$(p_1, ...p_n)$, the set of all possible streaks is denoted by $\mathcal{S}_P$. A streak $s \in \mathcal{S}_P$ is a *prominent streak* if it is not dominated by any streak in $\mathcal{S}_P$, i.e., $\nexists s'$ s.t. $s' \in \mathcal{S}_P$ and $s' \succ s$. The set of all prominent streaks in P is denoted by $\mathcal{PS}_P$. ■

Problem Statement: The *prominent streak discovery problem* is to, given a sequence P, produce $\mathcal{PS}_P$.

Figure 1 shows our running example of a 10-value sequence P=$(3, 1, 7, 7, 2, 5, 4, 6, 7, 3)$. There are 5 prominent streaks in P– $\langle [1, 10], 1\rangle$, $\langle [3, 10], 2\rangle$, $\langle [6, 10], 3\rangle$, $\langle [6, 9], 4\rangle$, $\langle [3, 4], 7\rangle$. For instance, $\langle [6, 9], 4\rangle$ is a prominent streak of minimal value 4, whose interval is from p_6 to p_9. $\langle [1, 10], 1\rangle$, the whole data sequence, is also a trivial prominent streak because no other streak can possibly dominate the sequence itself. The streak $\langle [8, 9], 6\rangle$ is an instance of non-prominent streaks because it is dominated by $\langle [3, 4], 7\rangle$.

1.2 Overview of the Solution

A brute-force method for discovering prominent streaks is not appealing. One can enumerate all possible streaks and decide if each streak is prominent by comparing it with every other streak. Given a sequence P with length n, there are $|\mathcal{S}_P|$=$\binom{n+1}{2}$ streaks in total. Thus the number of pair-wise streak comparison would be $\binom{|\mathcal{S}_P|}{2}$=$\frac{n^4+2n^3-n^2-2n}{8}$. Given a sequence of length 10000, the brute-force approach enumerates 10^8 streaks and performs 10^{16} comparisons. Many real-world sequences can be quite long. The sequence of daily closing prices for a stock with 40-year history has about 10000 values. A one-year usage log for a Web site has 8760 values at hourly interval.

Prominent streaks are in fact skyline points [5] in two dimensions– streak interval length $(r - l)$ and minimum value in the interval (v). A streak is a prominent streak (skyline point) if it is not dominated by any point, i.e., there exists no streak that has both longer interval and greater minimum value.

Based on this observation, our solution hinges upon the idea to separate the two steps of prominent streak discovery– *candidate streak generation* and *skyline operation* over candidate streaks. In candidate generation, we prune a large portion of non-prominent streaks without exhaustively considering all possible streaks. For skyline operation, we leverage efficient algorithms from the rich literature on this topic, e.g., [5, 21, 11, 14]. The effectiveness of pruning in the first step is critical to overall performance, because execution time of skyline algorithms increases superlinearly by the number of candidate points [5].

Candidate streak generation: We considered three methods with increasing pruning power in candidate generation– a baseline method, a non-linear LPS (local prominent streak)-based method, and a linear LPS-based method. The baseline method exhaustively enumerates $\mathcal{S}_P$, all possible streaks in a sequence P, by a nested-loop over the values in P. The sketch of this method is in Algorithm 1. It produces quadratic ($\frac{n(n+1)}{2}$) candidate streaks. We then propose the concept of *local prominent streak* (LPS) for substantially reducing the number of candidate streaks (Section 3). The intuition is, given a prominent streak s, there cannot be a super-sequence of s with greater or equal minimal value. In other words, s must be locally prominent as well. Hence we only need to consider LPSs as candidates. The algorithm sequentially scans the data sequence and maintains possible LPSs. We further make the observation that an LPS cannot have a preceding or succeeding data entry that is greater than or equal to its minimal value. Therefore we find the left-end (right-end) of an LPS when we find a data entry that is greater than its preceding (succeeding) entry. The non-linear LPS-based method finds a superset of LPSs as candidates, while the linear LPS-based method guarantees to find only LPSs.

Note that to couple candidate streak generation with skyline operation, Algorithm 1 maintains a dynamic skyline and updates it whenever a new candidate streak is produced. The updating procedure $skyline\_update$ is in Algorithm 2 and is introduced below.

Algorithm 1: Baseline Method

Input: Data sequence P=$(p_1, ..., p_n)$
Output: Prominent streaks $skyline$

```
1 skyline ← empty
2 for r = 1 to n do
3     min_value ← ∞
4     for l = r downto 1 do
5         min_value ← min(p_l, min_value)
6         s ← ⟨[l, r], min_value⟩ // candidate streak
7         skyline ← skyline_update(skyline, s)
```

Algorithm 2: Update Dynamic Skyline ($skyline\_update$)

Input: Dynamic skyline $skyline$, new candidate streak $s = \langle [l, r], v\rangle$
Output: Updated dynamic skyline $skyline$

```
1 Find the largest i in skyline s.t. v_i ≤ v
2 if s ≺ s_i or s ≺ s_{i+1} then
3     return skyline
4 while s ≻ s_i and i > 0 do
5     Delete s_i from skyline
6     i ← i − 1
7 Insert s into skyline
8 return skyline
```

Skyline operation: Our focus is not to compare various skyline algorithms. Many existing algorithms can be leveraged. What matters is the number of candidate streaks produced by the candidate generation step. This is also verified by our experiments which show that, under various skyline algorithms, the candidate streak generation methods in Section 3 perform and compare consistently.

We can use a sorting-based method for finding the skyline points in a two-dimensional space [5]. If the candidate streak generation step does not prune streaks effectively, we cannot hold all candidate streaks in memory. The memory overflow can be addressed by external-memory sorting.

Another approach is to progressively update a dynamic skyline

with candidate streaks, based on the nested-loop method in [5]. The outline of this approach is shown in Algorithm 2. We use $skyline$ to denote the dynamic skyline. When a new candidate streak s is generated, s is inserted into $skyline$ if it is not dominated by any point in $skyline$. The algorithm also checks if some points in $skyline$ are dominated by s and eliminates them from $skyline$.

The dominance relationship can be efficiently checked, given that the streaks have only two dimensions– interval length $(r - l)$ and minimum value (v). The key idea is that the lengths of streaks monotonically decrease as their minimal values increase (except that there can be identical points, i.e., streaks with equal lengths and equal minimal values.) Hence the streaks in $skyline$ are ordered by v (or by $r - l$). Suppose the candidate streak is $s = \langle [l, r], v \rangle$. We only have to find in $skyline$ a pivoting streak $s_i = \langle [l_i, r_i], v_i \rangle$ such that i is the largest index with $v_i \leq v$, i.e., $v_i \leq v < v_{i+1}$, then check its neighbors to determine the dominance relationship. For quickly finding s_i, we use a balanced binary search tree (BST) on v to store $skyline$. (Thus we call it BST-based skyline method.) We can prove that s is dominated by some points in $skyline$ if and only if s is dominated by s_i or s_{i+1}. Furthermore, if s dominates totally k streaks in $skyline$, then the k streaks are s_i, s_{i-1}, ..., s_{i-k+1}. (We omit the discussion of boundary cases, i.e., i=0 or i=$|skyline|$.)

In comparison with the sorting-based method, the above BST-based skyline method saves both memory space and execution time. It avoids memory overflow because the number of streaks in the dynamic skyline in most cases remains small enough to fit in memory. Hence no streak needs to be read from/written to secondary memory. The small size of dynamic skyline in real data is verified by our experiments in Section 5. After all, prominent streaks (and skyline points in general) are supposed to be minority, otherwise they cannot stand out to warrant further investigation. Furthermore, even if the dynamic skyline grows large, a method such as the block nested-loop based method in [5] can be applied to fall back on secondary memory. The small size of dynamic skyline also means small number of streak comparisons. Intuitively, given c candidate streaks, a fast comparison-based sorting algorithm (say quicksort) requires $O(c \log c)$ comparisons, while the BST-based method only requires $O(c \log s)$ comparisons, where s is the maximal size of the dynamic skyline during computation. Experiments in Section 5 show that s is typically much smaller than c.

1.3 Summary of Contributions and Outline

To summarize, this paper makes the following contributions:

- We defined the problem of prominent streak discovery. The simple concept is useful in many real-world applications. To the best of our knowledge, there has not been study along this line.
- We proposed the solution framework to separate *candidate streak generation* and *skyline operation* during prominent streak discovery. Under this framework, we designed efficient algorithms for candidate streak generation, based on the concept of local prominent streak. Both the non-linear LPS-based method (NLPS) and the linear LPS-based method (LLPS) produce substantially less candidate streaks than the quadratic number of candidates produced by a baseline method. LLPS further guarantees a linear number of candidate streaks.
- We conducted experiments over multiple real datasets. The results verified the effectiveness of our methods and showed orders of magnitude performance improvement over the baseline method. We also showed some insightful prominent streaks discovered from real data, to highlight the practicality of this work.

The rest of the paper is organized as follows. In Section 2 we review related work. Section 3 presents the NLPS and LLPS methods for candidate streak generation. Section 4 discusses how to adapt the algorithms to monitor prominent streaks when data sequence continuously grows. Experiment setup and results are reported in Section 5. Section 6 concludes the paper.

2. RELATED WORK

Data mining on sequence and time-series data has been an active area of research, where many techniques are developed for similarity search and subsequence matching in sequence and time-series databases [1, 8, 2, 26], finding sequential patterns [3, 20, 27, 15, 25], classification and clustering of sequence and time-series data [19, 13, 12, 18], biological sequence analysis [4, 17], etc. However, we are not aware of prior work on the prominent streak discovery problem proposed in this paper.

The skyline of a set of tuples is the subset of tuples that are not dominated by any tuple. A tuple dominates another tuple if it is equally good or better on every attribute and better on at least one attribute. Skyline query has been intensively studied over the last decade. Kung et al. [9] first proposed in-memory algorithms to tackle the skyline problem. Börzsönyi et al. [5] considered the problem in database context and integrated skyline operator into database system. They also invented a block-nested-loop algorithm (BNL) and extended the divide-and-conquer algorithm (DC) from [9]. An improvement of BNL is discussed in [6].

Progressive skyline algorithms optimize the efficiency in returning initial skyline points while producing more results progressively. Various algorithms developed along this line include the bitmap-based algorithm and the index-based algorithm [21], the nearest neighbor search algorithm [11], and the branch-and-bound skyline algorithm (BBS) [14]. Other variants of skyline queries have also been studied, including skyline cube which aims to answer skyline queries over any combination of dimensions [16, 24].

Jiang et al. [10] studied the problem of interval skyline queries on time-series. Given a set of time series and a time interval, they find the time series that are not dominated by others in the interval. A time series dominates another one if its value at every position is at least equal to the corresponding value in the other time series and it is at least larger at one position. The point-by-point equi-length interval comparison is clearly different from our problem.

The plateau of a time series is the time interval in which the vlaues are close to each other (within a given threshold) and are no smaller than the values outside the interval [22]. The plateau problem is not concerned about comparing different intervals.

Our techniques can be useful in disease outbreak detection, by identifying prominent streaks in time series of aggregated disease case counts. Previous works on outbreak detection focus on conventional data mining tasks such as clustering and regression [23]. The concept of prominent streaks has not been studied before.

3. DISCOVERING PROMINENT STREAKS FROM LOCAL PROMINENT STREAKS

For an n-element sequence P, the baseline method (Algorithm 1) produces $\frac{n(n+1)}{2}$ candidate streaks. In this section, based on the concept of *local prominent streak* (LPS) we propose the non-linear LPS-based (NLPS) and linear LPS-based (LLPS) methods. Both dramatically reduce the number of candidate streaks in practice. LLPS further guarantees only a linear number of candidate streaks.

3.1 Local Prominent Streak (LPS)

Definition 2 (Local Prominent Streak): Given a sequence of data values $P = (p_1, \cdots, p_n)$, we say a streak $s = \langle [l, r], v \rangle \in \mathcal{S}_P$ is a *local prominent streak* (LPS) or *locally prominent* if there does not

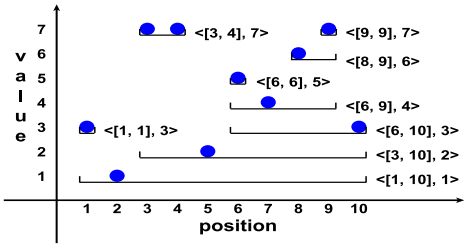


Figure 2: Local Prominent Streaks.

exist any other streak $s' = \langle [l', r'], v' \rangle \in \mathcal{S}_P$ such that $[l', r'] \supset [l, r]$ and $s' \succ s$. (I.e., there does not exist such s' that $[l', r'] \supset [l, r]$ and $v' \geq v$.) The symbol $\supset$ denotes the subsumption check between two intervals, i.e., $[l', r'] \supset [l, r]$ if and only if $l' \leq l \wedge r' > r$ or $l' < l \wedge r' \geq r$. We denote the set of local prominent streaks in sequence P as $\mathcal{LPS}_P$. ■

Figure 2 shows all the local prominent streaks found in our running example. All other streaks are not locally prominent. For example, $\langle [6, 8], 4 \rangle$ is not locally prominent since it is dominated by $\langle [6, 9], 4 \rangle$ and $[6, 9] \supset [6, 8]$. In the following we give several important properties of local prominent streaks.

Property 1: Every prominent streak is also a local prominent streak, i.e., $\mathcal{PS}_P \subseteq \mathcal{LPS}_P$. ■

Proof: Suppose there is a prominent streak that is not locally prominent, i.e., $\exists s \in \mathcal{PS}_P$ s.t. $s \notin \mathcal{LPS}_P$. By Definition 2, there exists some streak s' such that $[l', r'] \supset [l, r]$ and $s' \succ s$. That is contradictory to Definition 1 which says s cannot be dominated by any other streak. Therefore a streak cannot be prominent if it is not even locally prominent. ■

The above Property 1 is illustrated by Figure 2, as all the prominent streaks in Figure 1 also appear in Figure 2. However, the reverse of Property 1 does not hold– local prominent streaks are not necessarily prominent streaks. For example, $\langle [8, 9], 6 \rangle$ is an LPS but is dominated by $\langle [3, 4], 7 \rangle$ and therefore is not in Figure 1.

Lemma 1: Suppose $s = \langle [l, r], v \rangle$ and $s' = \langle [l', r'], v' \rangle$ are two different local prominent streaks in P, i.e., $s, s' \in \mathcal{LPS}_P, l \neq l'$ or $r \neq r'$. For any $k \in argmin_{i \in [l,r]} p_i$ and $k' \in argmin_{i \in [l',r']} p_i$, we have $k \neq k'$. I.e., $argmin_{i \in [l,r]} p_i \cap argmin_{i \in [l',r']} p_i = \phi$. ■

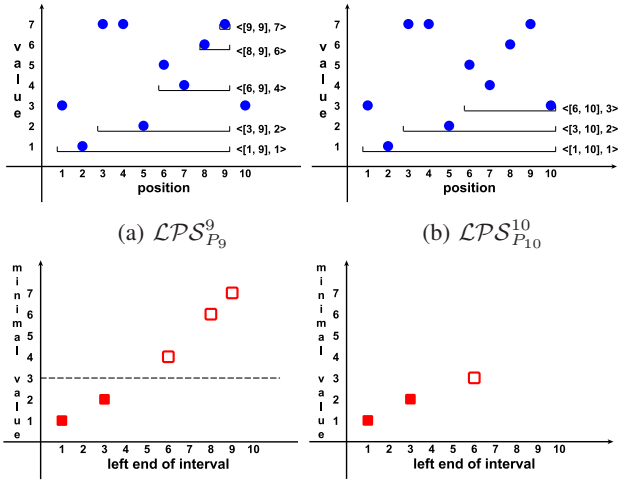
Proof: If $[l, r] \cap [l', r'] = \phi$, i.e., the two intervals do not overlap, it is obvious that $k \neq k'$. Now consider the case when $[l, r] \cap [l', r'] \neq \phi$, i.e., $l \leq l' \leq r$ or $l' \leq l \leq r'$. By definition of $argmin$, $p_k = v = min_{i \in [l,r]} p_i$ and $p_{k'} = v' = min_{i \in [l',r']} p_i$. Suppose there exist such k and k' that $k = k'$. Thus $v{=}v'{=}p_k$. By Definition 1, we have $p_i \geq v$ for every $i \in [l, r]$ and every $i \in [l', r']$. Since the two intervals $[l, r]$ and $[l', r']$ overlap, their combined interval corresponds to a new streak $s''{=}\langle [l, r] \cup [l', r'], v \rangle$. [1] It is clear $s'' \succ s$ and $s'' \succ s'$. That is a contradiction to the precondition that both s and s' are LPSs. Thus, this lemma holds. ■

Lemma 1 indicates that two different LPSs cannot reach their minimal values at the same position. Therefore each value position in sequence P can correspond to the minimal value of at most one LPS. What immediately follows is that there are at most n LPSs in an n-element sequence. Formally we have the following property.

Property 2: $|\mathcal{LPS}_P| \leq |P|$. ■

From Property 1 we know that $\mathcal{LPS}_P$ is a sufficient candidate set for $\mathcal{PS}_P$, i.e., we can guarantee to find all prominent streaks if we only consider local prominent streaks. Property 2 further shows how small $\mathcal{LPS}_P$ is and thus how good it is as a candidate set. Specifically, the size of $\mathcal{LPS}_P$ is at most $|P|$, the length of the sequence, in contrast to the all $\frac{|P|(|P|+1)}{2}$ possible streaks considered by the baseline method (Algorithm 1). Thus, $\mathcal{LPS}_P$ helps to prune most streaks from further consideration. In the following sections we present efficient algorithms for computing a superset of $\mathcal{LPS}_P$ and $\mathcal{LPS}_P$ itself exactly.

3.2 $\mathcal{LPS}_P^k$ and $\mathcal{LPS}_{P_k}^k$



(a) $\mathcal{LPS}_{P_9}^9$ (b) $\mathcal{LPS}_{P_{10}}^{10}$

(c) l-v plot of $\mathcal{LPS}_{P_9}^9$ (d) l-v plot of $\mathcal{LPS}_{P_{10}}^{10}$

Figure 3: From $\mathcal{LPS}_{P_9}^9$ to $\mathcal{LPS}_{P_{10}}^{10}$.

To facilitate our discussion, we first define a new notation, $\mathcal{LPS}_P^k$.

Definition 3: $\mathcal{LPS}_P^k$ is the set of local prominent streaks in P that end at position k, i.e., $\mathcal{LPS}_P^k{=}\{s | s \in \mathcal{LPS}_P$ and $s{=}\langle [l, k], v \rangle\}$. ■

There are two key components in the definition of $\mathcal{LPS}_P^k$. The first is the upper script k, which fixes the right end of every interval in the set. It is clear that $\mathcal{LPS}_P^1, \mathcal{LPS}_P^2, \ldots, \mathcal{LPS}_P^{|P|}$ is a natural partition of $\mathcal{LPS}_P$. We use this partition scheme in the design of our algorithms. Specifically, we show how each $\mathcal{LPS}_P^k$ in this partition is calculated in a sequential and progressive way.

The second key component in the definition of $\mathcal{LPS}_P^k$ is the lower script P, which provides the scope for local prominent streaks. By generalizing this component we define $\mathcal{LPS}_{P_k}^k$. We denote the sequence with the first k entries of P as P_k. Then $\mathcal{LPS}_{P_k}$ is the set of local prominent streaks with regard to sequence P_k (instead of P) and $\mathcal{LPS}_{P_k}^k$ are those LPSs in $\mathcal{LPS}_{P_k}$ that end at k. Due to the change of scope, $\mathcal{LPS}_{P_k}^k$ is a superset of $\mathcal{LPS}_P^k$. Formally, we have the following property.

Property 3: $\mathcal{LPS}_P^k \subseteq \mathcal{LPS}_{P_k}^k$. ■

Proof: Consider any streak $s \in \mathcal{LPS}_P^k$. By Definition 3, $s = \langle [l, k], v \rangle$ and $s \in \mathcal{LPS}_P$. Therefore by Definition 2, there does not exist any $s' = \langle [l', r'], v' \rangle$ in P such that $s' \succ s$ and $[l', r'] \supset [l, k]$. Since P_k is a prefix of P, i.e., the first k values in P, it follows that there does not exist any such s' in P_k either. Therefore $s \in \mathcal{LPS}_{P_k}^k$. ■

Consider the running example again. Figure 3(a) shows $\mathcal{LPS}_{P_9}^9$, including $\langle [1, 9], 1 \rangle, \langle [3, 9], 2 \rangle, \langle [6, 9], 4 \rangle, \langle [8, 9], 6 \rangle, \langle [9, 9], 7 \rangle$. As shown in Figure 2, $\mathcal{LPS}_P^9$ contains $\langle [6, 9], 4 \rangle, \langle [8, 9], 6 \rangle, \langle [9, 9], 7 \rangle$. Streaks $\langle [1, 9], 1 \rangle$ and $\langle [3, 9], 2 \rangle$ do not belong to $\mathcal{LPS}_P$, thus do not belong to $\mathcal{LPS}_P^9$, since they are locally dominated by $\langle [1, 10], 1 \rangle$

[1] The two intervals can overlap in four different ways. Thus $[l, r] \cup [l', r'] = [l, r]$ or $[l, r']$ or $[l', r]$ or $[l', r']$.

and $\langle[3,10],2\rangle$, respectively. By contrast, $\langle[1,9],1\rangle$ and $\langle[3,9],2\rangle$ are part of $\mathcal{LPS}^9_{P_9}$ because they are not locally dominated by any streak of P_9, which only contains the first 9 values of P.

3.3 Non-linear LPS Method

By Property 3 and the fact that $\mathcal{LPS}^1_P, \cdots, \mathcal{LPS}^{|P|}_P$ is a partition of $\mathcal{LPS}_P$, we have

$$\mathcal{LPS}_P = \bigcup_{1\le k\le |P|} \mathcal{LPS}^k_P \subseteq \bigcup_{1\le k\le |P|} \mathcal{LPS}^k_{P_k} \quad (1)$$

Thus, we can use $\bigcup_{1\le k\le |P|} \mathcal{LPS}^k_{P_k}$ as our candidate set for prominent streaks. Although its size can be greater than that of $\mathcal{LPS}_P$, in practice it does substantially reduce the size of candidate streaks, verified by the experimental results in Section 5.

Algorithm 3: Non-linear LPS Method (NLPS)

Input: Data sequence $P = (p_1, \cdots, p_n)$
Output: Prominent streaks $skyline$

```
1  skyline ← empty
2  for k = 1 to n do
3      Compute LPS^k_{P_k} by Algorithm 4
4      for each streak s in LPS^k_{P_k} do
5          skyline ← skyline_update(skyline, s)
```

Algorithm 4: Progressive Computation of $\mathcal{LPS}^k_{P_k}$

Input: $\mathcal{LPS}^{k-1}_{P_{k-1}}$ and p_k
Output: $\mathcal{LPS}^k_{P_k}$

```
// When it starts, stack lps consists of streaks in LPS^{k-1}_{P_{k-1}}.
1  pivot ← null
2  while ! lps.isempty() do
3      if lps.top().v < p_k then
4          break
5      else
6          pivot ← lps.pop()
7  if pivot == null then
8      lps.push(⟨[k,k], p_k⟩)
9  else
10     pivot.v ← p_k
11     lps.push(pivot)
// Now, lps contains all the streaks in LPS^k_{P_k}.
```

Along this line, Algorithm 3 presents the method to compute candidate streaks. Since the number of candidates may be super-linear to the length of data sequence, we call it the *non-linear LPS method* (NLPS). The algorithm iterates k from 1 to $|P|$, progressively computes $\mathcal{LPS}^k_{P_k}$ from $\mathcal{LPS}^{k-1}_{P_{k-1}}$ when the k-th element p_k is visited, and includes them into candidate streaks. The details of updating from $\mathcal{LPS}^{k-1}_{P_{k-1}}$ to $\mathcal{LPS}^k_{P_k}$ are in Algorithm 4, which is based on the following Lemma 2. For convenience of discussion, we first define the right-end extension of a streak and a streak set.

Definition 4: If $s = \langle[l,r],v\rangle$ is a streak in an n-element data sequence P and $r < n$, the right-end extension of s is streak $\langle[l, r+1], v'\rangle$, where $v' = \min\{v, p_{r+1}\}$. The extension of a streak set S is the set which consists of extensions of all the streaks in S. ■

Lemma 2: If $s_1{=}\langle[l,k],v_1\rangle \in \mathcal{LPS}^k_{P_k}$ and $l{\neq}k$, then the streak $s_2{=}\langle[l,k-1],v_2\rangle \in \mathcal{LPS}^{k-1}_{P_{k-1}}$. ■

Proof: First, note that $v_2{=}\min_{l_1\le i\le k-1} p_i$ and $v_1{=}\min\{v_2, p_k\}$. We prove by contradiction. Suppose $s_2{=}\langle[l_1,k{-}1],v_2\rangle{\notin}\mathcal{LPS}^{k-1}_{P_{k-1}}$. By Definition 3, $s_2{\notin}\mathcal{LPS}_{P_{k-1}}$. Further by Definition 2, there exists $s_3{=}\langle[l_3,r_3],v_3\rangle{\in}\mathcal{S}_{P_{k-1}}$ such that $[l_3,r_3] \supset [l_1,k-1]$ and $s_3 \succ s_2$. Given any $s{=}\langle[l,r],v\rangle{\in}\mathcal{S}_{P_{k-1}}$, we have $r \le k-1$. Therefore $r_3{=}k-1$, $l_3 < l_1$ and $v_3 \ge v_2$. The right-end extension of s_3 is $s_4{=}\langle[l_3,k],v_4\rangle$, where $v_4{=}\min\{v_3,p_k\}{\ge}\min\{v_2,p_k\}{=}v_1$. Therefore $s_4 \succ s_1$, which contradicts with the pre-condition that $s_1{\in}\mathcal{LPS}^k_{P_k}$. The property holds. ■

Lemma 2 indicates that, except $\langle[k,k],p_k\rangle$, for each streak in $\mathcal{LPS}^k_{P_k}$, its prefix streak is in $\mathcal{LPS}^{k-1}_{P_{k-1}}$. Hence, to produce $\mathcal{LPS}^k_{P_k}$, we only need to consider the right-end extension of $\mathcal{LPS}^{k-1}_{P_{k-1}}$. Beyond that, we only need to consider one extra streak $\langle[k,k],p_k\rangle$ since it may belong to $\mathcal{LPS}^k_{P_k}$ as well.

In order to articulate how to derive $\mathcal{LPS}^k_{P_k}$ from $\mathcal{LPS}^{k-1}_{P_{k-1}}$, we partition $\mathcal{LPS}^{k-1}_{P_{k-1}}$ into two disjoint sets, namely,

$$\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{<} = \{s | s = \langle[l,k-1],v\rangle \in \mathcal{LPS}^{k-1}_{P_{k-1}}, v < p_k\}, \quad (2)$$

$$\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{\ge} = \{s | s = \langle[l,k-1],v\rangle \in \mathcal{LPS}^{k-1}_{P_{k-1}}, v \ge p_k\}. \quad (3)$$

It is clear that $\mathcal{LPS}^{k-1}_{P_{k-1}}$ is the disjoint union of these two sets, i.e., $\mathcal{LPS}^{k-1}_{P_{k-1}} = \mathcal{LPS}^{k-1}_{P_{k-1}}{}^{<} \cup \mathcal{LPS}^{k-1}_{P_{k-1}}{}^{\ge}$, and $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{<} \cap \mathcal{LPS}^{k-1}_{P_{k-1}}{}^{\ge} = \phi$. Use the running example again. For $\mathcal{LPS}^9_{P_9}$ in Figure 3(a), since $p_{10} = 3$, the two sets are $\mathcal{LPS}^9_{P_9}{}^{<} = \{\langle[1,9],1\rangle, \langle[3,9],2\rangle\}$, $\mathcal{LPS}^9_{P_9}{}^{\ge} = \{\langle[6,9],4\rangle, \langle[8,9],6\rangle, \langle[9,9],7\rangle\}$.

We consider how to extend streaks in $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{<}$ and $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{\ge}$, respectively. For simplicity of presentation, we omit the formal proofs when we make various statements below.

- $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{<}$: We use $S1$ to denote the right-end extension of $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{<}$. Since every streak in $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{<}$ has a minimal value less than p_k, the corresponding extended new streak has the same minimal value. Hence all the new streaks belong to $\mathcal{LPS}^k_{P_k}$. For the running example, corresponding to $\mathcal{LPS}^9_{P_9}{}^{<}$, we have $S1 = \{\langle[1,10],1\rangle, \langle[3,10],2\rangle\}$.
- $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{\ge}$: We use $S2$ to denote the right-end extension of $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{\ge}$. Since every streak in $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{\ge}$ has a minimal value greater than or equal to p_k, the minimal value of every streak in $S2$ equals p_k. Hence, the longest streak in $S2$, denoted as $S2^*$, dominates all other streaks in $S2$ and it is the only streak in $S2$ that belongs to $\mathcal{LPS}^k_{P_k}$. Furthermore, since every streak in $S2$ has the same r value (the right end of the interval), i.e., k, $S2^*$ is the streak with the minimal l value (the left end of the interval) in $S2$. Clearly there cannot be another streak in $S2$ with the same length. For the running example, corresponding to $\mathcal{LPS}^9_{P_9}{}^{\ge}$, we have $S2 = \{\langle[6,10],3\rangle, \langle[8,10],3\rangle, \langle[9,10],3\rangle\}$. The longest streak in $S2$ is $\langle[6,10],3\rangle$.
- $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{\ge} = \phi$: If $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{\ge}$ is empty, a new streak $\langle[k,k],p_k\rangle$ belongs to $\mathcal{LPS}^k_{P_k}$. (Otherwise, it is dominated by $S2^*$.)

The above discussion is captured by the following Property 4.

Property 4: $\mathcal{LPS}^k_{P_k} = S1 \cup \{S2^*\}$ if $S2 \neq \phi$ and $\mathcal{LPS}^k_{P_k} = S1 \cup \{\langle[k,k],p_k\rangle\}$ if $S2 = \phi$. ■

We use Figure 3 to explain the above procedure of producing $\mathcal{LPS}^k_{P_k}$ from $\mathcal{LPS}^{k-1}_{P_{k-1}}$. Figure 3(a) and 3(b) show $\mathcal{LPS}^9_{P_9}$ and $\mathcal{LPS}^{10}_{P_{10}}$, respectively. Figure 3(c) and 3(d) also show $\mathcal{LPS}^9_{P_9}$ and $\mathcal{LPS}^{10}_{P_{10}}$, by using a different presentation– l-v plot. All the streaks $\langle[l,r],v\rangle$ in $\mathcal{LPS}^{k-1}_{P_{k-1}}$ share the same value of r, which is $k-1$. Therefore we plot the streaks by l (x-axis) and v (y-axis). In Figure 3(c), the 5 points represent the 5 streaks in $\mathcal{LPS}^9_{P_9}$: $\langle[1,9],1\rangle$, $\langle[3,9],2\rangle$, $\langle[6,9],4\rangle$, $\langle[8,9],6\rangle$, $\langle[9,9],7\rangle$. The dotted line represents the 10-th data entry $p_{10} = 3$. It bisects $\mathcal{LPS}^9_{P_9}$ into $\mathcal{LPS}^9_{P_9}{}^{\ge}$ (3 hollow points above the line) and $\mathcal{LPS}^9_{P_9}{}^{<}$ (2

filled points below the line). We produce new candidate streaks $\mathcal{LPS}^{10}_{P_{10}}$ by extending the right ends of streaks in $\mathcal{LPS}^{9}_{P_9}$ to 10. The streaks extended from $\mathcal{LPS}^{9}_{P_9}{}^{<}$ all belong to $\mathcal{LPS}^{10}_{P_{10}}$. They are the 2 filled points in Figure 3(d), corresponding to $\langle[1, 10], 1\rangle$ and $\langle[3, 10], 2\rangle$. Among the streaks extended from $\mathcal{LPS}^{9}_{P_9}{}^{\geq}$, only the one with the smallest l (the longest one) belongs to $\mathcal{LPS}^{10}_{P_{10}}$. It is the hollow point in Figure 3(d), corresponding to $\langle[6, 10], 3\rangle$. Hence $\mathcal{LPS}^{10}_{P_{10}}$=$\{\langle[1, 10], 1\rangle, \langle[3, 10], 2\rangle, \langle[6, 10], 3\rangle\}$.

The details of the algorithm are shown in Algorithm 4. We use a stack lps to maintain $\mathcal{LPS}^{k}_{P_k}$. Since the streaks $\langle[l, r], v\rangle$ in $\mathcal{LPS}^{k}_{P_k}$ have the same r value which equals k, we do not need to store r in lps. Hence each item in lps has two data attributes, v and l. The items in the stack are ordered by v (and l, since their v and l values both strictly monotonically increase).[2] In fact, Figure 3(c) and 3(d) visualize all items in lps, before and after p_{10} is encountered, respectively. In each figure, the leftmost point denotes the bottom of the stack (with the smallest v), while the rightmost point denotes the top of the stack (with the largest v). After data entries p_1, ..., p_{k-1} are encountered, lps contains $\mathcal{LPS}^{k-1}_{P_{k-1}}$. Given data entry p_k, we popped from the stack all the streaks whose v values are greater than or equal to p_k. Among the popped streaks, the leftmost one (with the smallest l and v) is pushed back into the stack, with v value replaced by p_k and r extended from $k-1$ to k. (Again, the r value is not explicitly stored in the stack.) If no streak was popped, then $\langle[k, k], p_k\rangle$ is pushed into the stack. The remaining streaks in the original stack are kept, with their v and l values unchanged and r extended from $k-1$ to k.

Algorithm 3 computes candidate streaks for an n-element sequence P. It invokes Algorithm 4 n times.[3] In each invocation, exactly 1 item is pushed into the stack. Therefore in total there are n insertions and thus at most n deletions. Hence, the amortized time complexity of Algorithm 4 is $O(1)$.

In each iteration of Algorithm 3, we compute $\mathcal{LPS}^{k}_{P_k}$ and include them into candidate streaks. Thus, for an n-element sequence, the total number of candidate streaks considered is $\sum_{k=1}^{n} |\mathcal{LPS}^{k}_{P_k}|$. In the worst case, we may have a strictly increasing sequence and the candidate streaks include all possible streaks. This is as bad as the exhaustive baseline method in Algorithm 1. For example, given sequence $\langle 10, 20, 30\rangle$, we have $\mathcal{LPS}^{1}_{P_1} = \{\langle[1, 1], 10\rangle\}$, $\mathcal{LPS}^{2}_{P_2} = \{\langle[1, 2], 10\rangle, \langle[2, 2], 20\rangle\}$ and $\mathcal{LPS}^{3}_{P_3} = \{\langle[1, 3], 10\rangle, \langle[2, 3], 20\rangle, \langle[3, 3], 30\rangle\}$.

3.4 Linear LPS Method

Now we present the linear LPS (LLPS) method (Algorithm 5), which guarantees to produce a linear number of candidate streaks even in the worst case. Similar to Algorithm 3, this method iterates through the data sequence and computes $\mathcal{LPS}^{k}_{P_k}$ from $\mathcal{LPS}^{k-1}_{P_{k-1}}$ when the k-th data entry is encountered, for k from 1 to n. However, different from Algorithm 3, it also computes $\mathcal{LPS}^{k-1}_{P}$ from $\mathcal{LPS}^{k-1}_{P_{k-1}}$. Computation of both $\mathcal{LPS}^{k}_{P_k}$ and $\mathcal{LPS}^{k-1}_{P}$ is done in Algorithm 6, which is a simple extension of Algorithm 4. It is worth noting that, since P_n=P, $\mathcal{LPS}^{n}_{P}$ and $\mathcal{LPS}^{n}_{P_n}$ are identical.

To produce $\mathcal{LPS}^{k-1}_{P}$ from $\mathcal{LPS}^{k-1}_{P_{k-1}}$ given the k-th entry p_k, Algorithm 6 is based on the following Property 5. Due to space limitations, we omit the formal proof, but explain its intuition as follows. Recall that the minimal value of any streak in $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{\geq}$ (Equation (3)) is not smaller than p_k. It follows that if the minimal value of a streak in $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{\geq}$ is greater than p_k, the streak cannot grow into a longer local prominent streak without changing the minimal value. Hence, the streak itself is a local prominent streak. To summarize, $\mathcal{LPS}^{k-1}_{P}$ is the same as $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{\geq}$. The only exception is the longest streak in $\mathcal{LPS}^{k-1}_{P_{k-1}}{}^{\geq}$, i.e., the streak with the smallest l and thus the smallest minimal value v. If its minimal value is equal to p_k, then it does not belong $\mathcal{LPS}^{k-1}_{P}$, because it can be right-extended and included in $\mathcal{LPS}^{k'}_{P}$ for some $k' \geq k$.

Property 5: Given an n-entry sequence P, for any position $1<k\leq n$, $\mathcal{LPS}^{k-1}_{P} = \{s | s = \langle[l, k-1], v\rangle \in \mathcal{LPS}^{k-1}_{P_{k-1}}{}^{\geq}$ and $v > p_k\}$. ■

Continue the running example. We have $\mathcal{LPS}^{9}_{P} = \mathcal{LPS}^{9}_{P_9}{}^{\geq} = \{\langle[6, 9], 4\rangle, \langle[8, 9], 6\rangle, \langle[9, 9], 7\rangle\}$. Note that $\mathcal{LPS}^{9}_{P_9}{}^{\geq}$ and $\mathcal{LPS}^{9}_{P}$ are identical because the minimal values for the streaks in $\mathcal{LPS}^{9}_{P_9}{}^{\geq}$ are all greater than p_{10}.

Similar to Algorithm 4, Algorithm 6 has an amortized time complexity of $O(1)$. However, in LLPS we only need to consider the streaks in $\mathcal{LPS}^{k-1}_{P}$ as candidates. Consequently, LLPS reduces the total number of candidate streaks to $\sum_{k=1}^{n} |\mathcal{LPS}^{k}_{P}|$, i.e., $|\mathcal{LPS}_P|$ (Equation (1)). By Property 2, $|\mathcal{LPS}_P|$ is n at most, thus LLPS guarantees to produces only a linear number of candidate streaks even in worst case.

Algorithm 5: Linear LPS Method(LLPS)

Input: Data sequence $P = (p_1, \ldots p_n)$
Output: Prominent streaks $skyline$

```
1 skyline ← empty
2 for k = 1 to n do
3     Compute LPS_P^{k-1} and LPS_{P_k}^k by Algorithm 6
4     for each streak s in LPS_P^{k-1} do
5         skyline ← skyline_update(skyline, s)
6 LPS_P^n ← LPS_{P_n}^n
7 for each streak s in LPS_P^n do
8     skyline ← skyline_update(skyline, s)
```

Algorithm 6: Computing $\mathcal{LPS}^{k-1}_{P}$ and $\mathcal{LPS}^{k}_{P_k}$

Input: $\mathcal{LPS}^{k-1}_{P_{k-1}}$ and p_k
Output: $\mathcal{LPS}^{k-1}_{P}$ and $\mathcal{LPS}^{k}_{P_k}$

// Insert the following line before Line 1 in Algorithm 4.
$\mathcal{LPS}^{k-1}_{P} \leftarrow \phi$

// Insert the following two lines after Line 6 in Algorithm 4, in the same **else** branch as Line 6.
if $pivot.v > p_k$ **then**
    $\mathcal{LPS}^{k-1}_{P} \leftarrow \mathcal{LPS}^{k-1}_{P} \cup \{pivot\}$

4. MONITORING PROMINENT STREAKS

One desirable property of a prominent streak discovery algorithm is the capability of monitoring new data entries as the sequence grows continuously and always keeping the prominent streaks up-to-date. For example, a network administrator may check the prominent streaks in the network traffic of a Web server till any particular moment. Formally, given a continuously growing data sequence P (such as a data stream), the k-th data entry that has just come is denoted by p_k and the sequence so far is denoted by P_k. At this moment, if the user requests $\mathcal{PS}_{P_k}$, the prominent streaks of P_k, our method should efficiently discover them.

The BST-based method progressively updates the dynamic skyline with new candidate streaks, thus can be applied for monitoring prominent streaks without modification.

[2] We omit the proof of monotonicity.

[3] With regard to the first data element p_1, $\langle[1, 1], p_1\rangle$ is pushed into the stack. It is the only prominent streak and local prominent streak for P_1.

name	length	# prominent streaks	description
Gold	1074	137	Daily morning gold price in US dollars, 01/1985-03/1989.
River	1400	93	Mean daily flow of Saugeen River near Port Elgin, 01/1988-12/1991.
Melb1	3650	55	The daily minimum temperature of Melbourne, Australia, 1981-1990.
Melb2	3650	58	The daily maximum temperature of Melbourne, Australia, 1981-1990.
Wiki1	4896	58	Hourly traffic to `en.wikipedia.org/wiki/Main_page`, 04/2010-10/2010.
Wiki2	4896	51	Hourly traffic to `en.wikipedia.org/wiki/Lady_gaga`, 04/2010-10/2010.
Wiki3	4896	118	Hourly traffic to `en.wikipedia.org/wiki/Inception_(film)`, 04/2010-10/2010.
SP500	10136	497	S&P 500 index, 06/1960-06/2000.
HPQ	12109	232	Closing price of HPQ in NYSE for every trading day, 01/1962-02/2010.
IBM	12109	198	Closing price of IBM in NYSE for every trading day, 01/1962-02/2010.
AOL	132480	127	Number of queries sent to AOL search engine in every minute over three months.
WC98	7603201	286	Number of requests to World Cup 98 web site in every second, 04/1998-07/1998.

Table 1: Data Sequences Used in Experiments.

With regard to candidate streak generation, all three methods (baseline, NLPS, LLPS) use one-pass sequential scan of the data sequence, therefore they all naturally fit into the monitoring scenario. Specifically, the new data point p_k corresponds to the next iteration of the outer loop in Algorithm 1, 3, and 5. The baseline method exhaustively lists all streaks ending at p_k and updates the skyline with these streaks. The NLPS method updates $\mathcal{LPS}^{k-1}_{P_{k-1}}$ to $\mathcal{LPS}^{k}_{P_k}$, and updates the skyline with the streaks in $\mathcal{LPS}^{k}_{P_k}$.

The adaptation of LLPS is a bit more complex, as shown in Algorithm 7. This algorithm records the last position when the user requested the prominent streaks. When p_k arrives, $\mathcal{LPS}^{k-1}_{P}$ and $\mathcal{LPS}^{k}_{P_k}$ are dynamically computed by Algorithms 6. The skyline is updated with the candidate streaks in $\mathcal{LPS}^{k-1}_{P}$, only if $\mathcal{PS}_{P_{k-1}}$ was not requested by the user when p_{k-1} was visited. Note that if $\mathcal{PS}_{P_{k-1}}$ was requested, the skyline has already been updated with the streaks in $\mathcal{LPS}^{k-1}_{P_{k-1}}$. Since $\mathcal{LPS}^{k-1}_{P} \subseteq \mathcal{LPS}^{k-1}_{P_{k-1}}$, we do not need to update the skyline with $\mathcal{LPS}^{k-1}_{P}$ again. Finally, if the user requests $\mathcal{PS}_{P_k}$, the skyline has to be updated with $\mathcal{LPS}^{k}_{P_k}$ since all the local prominent streaks (with regard to P_k) ending at p_k must be considered. In Section 5 we will show the significant superiority of this adaptation of LLPS over other methods.

Note that this algorithm degrades to NLPS (Algorithm 3) if the user requests the prominent streaks at every data entry. On the other hand, if the prominent streaks are only requested at p_n, i.e., the last entry in the sequence, it becomes the same as LLPS (Algorithm 5).

Algorithm 7: Continuous Monitoring of Prominent Streaks

Input: The new data entry p_k
Compute $\mathcal{LPS}^{k-1}_{P}$ and $\mathcal{LPS}^{k}_{P_k}$ by Algorithms 6
if $last\_requested\_position < k-1$ **then**
  for each streak s in $\mathcal{LPS}^{k-1}_{P}$ **do**
    $skyline \leftarrow sklyine\_update(skyline, s)$
if $\mathcal{PS}_{P_k}$ is requested **then**
  for each streak s in $\mathcal{LPS}^{k}_{P_k}$ **do**
    $skyline \leftarrow sklyine\_update(skyline, s)$
  $last\_requested\_position \leftarrow k$
  // Now, $skyline$ contains all prominent streaks in $\mathcal{PS}_{P_k}$

5. EXPERIMENTS

The algorithms were implemented in Java. The experiments were conducted on a computer with 2.26GHz Intel Core 2 Duo CPU under Windows 7. The limit on the heap size of Java Virtual Machine (JVM) was set at 512MB.

We used multiple real-world datasets, including time series data library[4], Wikipedia traffic statistics dataset[5], NYSE exchange data[6], AOL search engine log[7], and FIFA World Cup 98 web site access log[8]. These datasets cover a variety of application scenarios, including meteorology, hydrology, finance, web log, and network traffic. Table 1 shows the information of 12 data sequences from these data sets that we used in experiments. For each data sequence, we list its name, length, and the number of prominent streaks in the sequence. Each data sequence was stored in a data file.

[4]http://robjhyndman.com/TSDL/

[5]http://dammit.lt/wikistats/

[6]http://infochimps.com/datasets/daily-1970-current-open-close-hi-low-and-volume-nyse-exchange-up

	Baseline	NLPS	LLPS
Gold	5.77×10^5	6.04×10^4	1.05×10^3
River	9.81×10^5	2.18×10^4	1.33×10^3
Melb1	6.66×10^6	4.47×10^4	3.50×10^3
Melb2	6.66×10^6	4.28×10^4	3.49×10^3
Wiki1	1.20×10^7	7.16×10^4	4.79×10^3
Wiki2	1.20×10^7	5.77×10^4	4.75×10^3
Wiki3	1.20×10^7	7.31×10^4	4.70×10^3
SP500	5.14×10^7	1.69×10^6	9.98×10^3
HPQ	7.33×10^7	5.24×10^5	1.08×10^4
IBM	7.33×10^7	6.97×10^5	1.13×10^4
AOL	8.78×10^9	3.53×10^6	1.20×10^5
WC98	2.89×10^{13}	1.78×10^8	6.69×10^6

Table 2: Number of Candidate Streaks.

Examples of Interesting Prominent Streaks Discovered:

From 1985 to 1989, there had been more than one thousand consecutive trading days with morning gold price greater than $300. During this period, there had been a streak of four hundred days with price more than $400, though the $500 price only lasted two days at most.

In Melbourne, Australia, during the years between 1981 and 1990, the weather had been pleasant. There had been more than two thousand days with minimal temperature above zero, and the streak was not ending. (We do not have data beyond 1990.) The longest streak during which the temperature hit above 35 degrees Celsius is six days. It was in the summer of the year 1981.

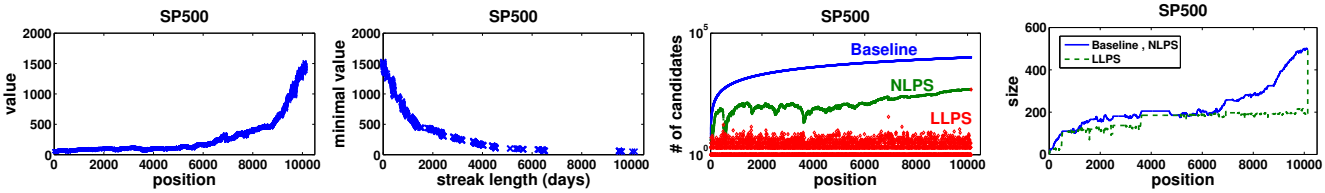
More than half of the prominent streaks we found in the traffic data of the Lady Gaga Wikipedia page were around September 12th, when she became a big winner in the MTV Video Music Awards (VMA) 2010. During that time, the page had been visited by at least 2000 people in every hour for almost four days.

Number of Candidate Streaks:

The three algorithms for candidate streak generation, namely Baseline (Algorithm 1), NLPS (Algorithm 3), and LLPS (Algorithm 5), differ by the ways they produce candidates and thus the numbers of produced candidates. Table 2 shows the total number of candidate streaks considered by each algorithm on each data sequence. The baseline algorithm produces an extremely large number of candidates since it enumerates all possible streaks, e.g., $\binom{7603202}{2}$=2.89×10^{13} for WC98. By contrast, NLPS only needs to

[7]http://gregsadetsky.com/aol-data/

[8]http://ita.ee.lbl.gov/html/contrib/WorldCup.html



(a) Data Sequence (b) Prominent Streaks (c) Number of Candidate Streaks (d) Size of Dynamic Skyline

Figure 4: Detailed Results on SP500.

data sequence	Baseline	NLPS	LLPS
Gold	78	40	9
River	78	15	3
Melb1	390	22	12
Melb2	387	21	15
Wiki1	711	28	15
Wiki2	711	18	15
Wiki3	689	40	16
SP500	4717	599	21
HPQ	6099	165	18
IBM	5079	209	22
AOL	446622	546	78
WC98	>1 hour	27477	3404

Table 3: Execution Time (in Milliseconds).

consider $\bigcup_{1\le k\le |P|} \mathcal{LPS}^k_{P_k}$, which is a superset of the real prominent streaks $\mathcal{PS}_P$ but a much smaller subset of all possible streaks $\mathcal{S}_P$. For instance, the number of candidate streaks by NLPS is 1.78×10^8 for WC98, which is 5 orders of magnitude smaller than what Baseline considers. LLPS further significantly educes the number of candidates by only considering LPSs. For example, there are 6.69×10^6 LPSs in WC98, which is about 30 times smaller than 1.78×10^8. Note that the number of LPSs for LLPS is bounded by sequence length (Property 2), which is verified by Table 2.

Execution Time:

The number of candidate streaks directly determines the efficiency of our algorithms. In Table 3 we report the execution time of our algorithms using the three candidate streak generation methods (Baseline, NLPS, LLPS), for all 12 data sequences. For skyline operation, we implemented the sorting-based, external-memory sorting-based, and BST-based skyline methods mentioned in Section 1. Under these different skyline methods, Baseline, NLPS, and LLPS perform and compare consistently. Therefore in Table 3 we only report the results for implementations based on the BST-based skyline method, due to space limitations. The reported execution time is in milliseconds and is the average of five runs.

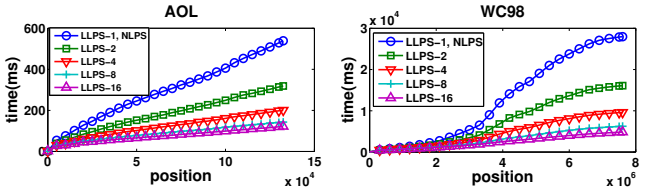
We excluded data loading time, i.e., the time spent on just reading each data file. This is because data loading time is dominated by processing time of the algorithms once the data file gets large. In our experiments, WC98 cost 1 second to load while the loading time of all other datasets was below 30ms.

In Table 3 we use '>1 hour' to denote the execution time when an algorithm could not finish within one hour (i.e., 3600000ms). This lower bound is sufficient in showing the performance difference of the various algorithms.

With regard to the comparison of Baseline, NLPS, and LLPS, it is clear from Table 3 that LLPS outperforms NLPS, and both NLPS and LLPS are far more efficient than Baseline. This is exactly due to the large gap in the number of candidate streaks (shown in Table 2), which in turn determines the number of comparisons performed during skyline operations.

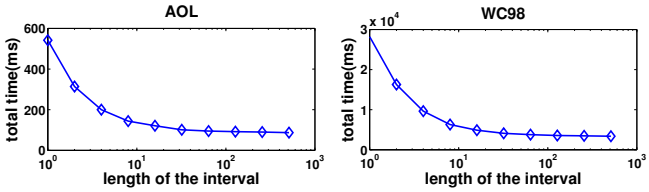
A Closer Look:

To have a better understanding of the experimental results, we take a close look at the SP500 data sequence. Similar observations are made on other data sequences, hence we omit the analysis of other sequences. Figure 4(a) shows the data sequence itself. We see that the sequence is almost monotonically increasing at the coarse grain level. Due to that, the number of prominent streaks found in SP500 (497, as shown in Table 1) is the most among all the data sequences. We also visualize the prominent streaks in SP500 in Figure 4(b), where the x-axis is for interval length and the y-axis is for minimal value in the interval.



(a) AOL (b) WC98

Figure 5: Cumulative Execution Time at Various Positions, for Different Reporting Frequencies.



(a) AOL (b) WC98

Figure 6: Total Execution Time by Reporting Frequencies.

In Table 2 we have seen the huge difference among Baseline, NLPS and LLPS in total number of candidate streaks. These three algorithms all generate candidates progressively. Therefore in Figure 4(c) we show, for each algorithm, the number of new candidate streaks produced at every value position of the data sequence. The figure clearly shows the superiority of LLPS since it always generates orders of magnitude less candidates at each position.

The BST-based skyline method maintains a dynamic skyline, as a binary search tree, in memory. The size of this tree affects the efficiency of tree operations, such as inserting and deleting a streak. Figure 4(d) shows the size of the dynamic skyline along the sequence of SP500 by each algorithm. The curves for Baseline and NLPS overlap since they both store $\mathcal{PS}_{P_k}$, at every position k, in the dynamic skyline. In contrary, LLPS does not need to store some streaks in $\mathcal{PS}_{P_k}$, hence the tree size is much smaller than that for Baseline/NLPS when the sequence is almost constantly growing in the second half of SP500.

Monitoring Prominent Streaks:

In Section 4 we discussed how to monitor the prominent streaks as a data sequence evolves and new data values come. The adaptation of LLPS for monitoring purpose was shown in Algorithm 7. This algorithm can control at which positions the prominent streaks (so far) need to be reported.

Take AOL and WC98 as examples. Figure 5 shows the execution time of Algorithm 7). The x-axis represents the sequence po-

sition, and the y-axis is for the total execution time by that position. There are five curves in each figure, corresponding to five different frequencies of reporting prominent streaks. For instance, LLPS-1 means that, whenever a new data entry comes, all the prominent streaks so far are reported; LLPS-16 means the prominent streaks are requested at every 16 data entries. As discussed in Section 4, LLPS-1 is identical to NLPS (Algorithm 3), and LLPS-n is identical to LLPS (Algorithm 5), where n is the sequence length when it does not evolve anymore. Figures 5(a) and 5(b) clearly show that the total execution time of LLPS-i increases as the reporting frequency increases (i.e., reporting interval i decreases). Figures 6(a) and 6(b) further show how the total execution time changes along different reporting intervals. We can see that the execution time drops rapidly at the beginning and quickly reaches near-optimal value even when the frequency is still pretty high (e.g., reporting the prominent streaks at every 16 entries.)

6. CONCLUSION

In this paper, we study the problem of discovering prominent streaks in sequence data. A prominent streak is a long consecutive subsequence consisting of only large (small) values. We propose efficient methods based on the concept of local prominent streak (LPS). We prove that prominent streaks are a subset of LPSs and the number of LPSs is less than the length of a data sequence. Our linear LPS-based method guarantees to consider only local prominent streaks, thus achieving significant reduction in candidate streaks. The results of experiments over multiple real datasets verified the effectiveness of the proposed methods.

Acknowledgements: We thank Jun Yang for discussion of the initial ideas of this paper when Chengkai Li and Jun Yang were both visiting HP Labs in Beijing, China in the summer of 2010.

7. REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. *Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[2] R. Agrawal, K. ip Lin, H. S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *VLDB*, pages 490–501, 1995.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 3–14, 1995.

[4] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[5] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering*, pages 421–430, 2001.

[6] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *Proceedings of the International Conference on Data Engineering*, pages 717–719, 2003.

[7] S. Cohen, C. Li, J. Yang, and C. Yu. Computational journalism: A call to arms to database researchers. In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 148–151, 2011.

[8] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, 1994.

[9] H.T.Kung, F.Luccio, and F.P.Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*, 22(4):469 – 476, 1975.

[10] B. Jiang and J. Pei. Online interval skyline queries on time series. In *Proceedings of the 25th International Conference on Data Engineering*, pages 1036–1047, 2009.

[11] D. Kossmann, F. Ramsak, and S. 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, 2002.

[12] T. W. Liao. Clustering of time series data–a survey. *Pattern Recognition*, 38(11):1857 – 1874, 2005.

[13] T. Oates, L. Firoiu, and P. Cohen. Clustering time series with hidden markov models and dynamic time warping. In *Proceedings of the IJCAI-99 Workshop on Neural, Symbolic and Reinforcement Learning Methods for Sequence Learning*, pages 17–21, 1999.

[14] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems (TODS)*, 30(1):41–82, 2005.

[15] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Mining sequential patterns by pattern-growth: the prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440, 2004.

[16] J. Pei, Y. Yuan, X. Lin, W. Jin, M. Ester, Q. Liu, W. Wang, Y. Tao, J. X. Yu, and Q. Zhang. Towards multidimensional subspace skyline analysis. *ACM Trans. Database Syst.*, 31(4):1335–1381, 2006.

[17] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[18] Y. Shin and D. Fussell. Parametric kernels for sequence data analysis. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1047–1052, 2007.

[19] P. Smyth. Clustering sequences with hidden Markov models. In *Advances in neural information processing systems*, 1997.

[20] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. *Advances in Database Technology (EDBT)*, pages 1–17, 1996.

[21] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 301–310, 2001.

[22] M. Wang and X. Wang. Finding the plateau in an aggregated time series. In *Advances in Web-Age Information Management (WAIM)*, pages 325–336, 2006.

[23] W.-K. Wong. *Data mining for early disease outbreak detection*. PhD thesis, Pittsburgh, PA, USA, 2004.

[24] T. Xia and D. Zhang. Refreshing the sky: the compressed skycube with efficient support for frequent updates. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 491–502, 2006.

[25] X. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large datasets. In *Proceedings of SIAM International Conference on Data Mining*, pages 166–177, 2003.

[26] B. Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings of the 14th International Conference on Data Engineering*, pages 201–208, 1998.

[27] M. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1):31–60, 2001.