Discovering General Prominent Streaks in Sequence Data

Article *in* ACM Transactions on Knowledge Discovery from Data · June 2014

DOI: 10.1145/2601439

5 authors, including:

Gensheng Zhang
University of Texas at Arlington
10 PUBLICATIONS **29** CITATIONS
SEE PROFILE

Ping Luo
HP Inc.
77 PUBLICATIONS **828** CITATIONS
SEE PROFILE

Min Wang
Fudan University
121 PUBLICATIONS **2,536** CITATIONS
SEE PROFILE

Discovering General Prominent Streaks in Sequence Data

GENSHENG ZHANG, The University of Texas at Arlington
XIAO JIANG, Shanghai Jiao Tong University
PING LUO, HP Labs China
MIN WANG, Google Research
CHENGKAI LI, The University of Texas at Arlington

9

This article 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, such as consecutive games of outstanding performance in sports, consecutive hours of heavy network traffic, and consecutive days of frequent mentioning of a person in social media. Prominent streak discovery provides insightful data patterns for data analysis in many real-world applications and is an enabling technique for computational journalism. Given its real-world usefulness and complexity, the research on prominent streaks in sequence data opens a spectrum of challenging problems.

A baseline approach to finding prominent streaks is a quadratic algorithm that exhaustively enumerates all possible streaks and performs pairwise streak dominance comparison. For more efficient methods, we make the observation that prominent streaks are in fact skyline points in two dimensions—streak interval length and minimum value in the interval. Our solution thus hinges on 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 the brute-force baseline method. We develop efficient algorithms based on the concept of LPS. The nonlinear local prominent streak (NLPS)-based method considers a superset of LPSs as candidates, and the linear local prominent streak (LLPS)-based method further guarantees to consider only LPSs. The proposed properties and algorithms are also extended for discovering general top-k, multisequence, and multidimensional prominent streaks. 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

Additional Key Words and Phrases: Computational journalism, sequence database, time series database, skyline query

---

This material is based on work partially supported by NSF Grants IIS-1018865 and CCF-1117369, and HP Labs Innovation Research Award. 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 the funding agencies.
Authors' addresses: G. Zhang, Department of Computer Science and Engineering, The University of Texas at Arlington; email: gensheng.zhang@mavs.uta.edu; X. Jiang, Department of Computer Science, Shanghai Jiao Tong University; email: showbufire@gmail.com; P. Luo, Institute of Computing Technology, CAS; email: luop@ict.ac.cn; M. Wang, whose bulk of research was done at HP Labs China, is currently affiliated with Google Research; email: minw83@gmail.com; C. Li, Department of Computer Science and Engineering, The University of Texas at Arlington; email: cli@uta.edu.

DOI: http://dx.doi.org/10.1145/2601439

ACM Reference Format:
Gensheng Zhang, Xiao Jiang, Ping Luo, Min Wang, and Chengkai Li. 2014. Discovering general prominent streaks in sequence data. ACM Trans. Knowl. Discov. Data 8, 2, Article 9 (May 2014), 37 pages.
DOI: http://dx.doi.org/10.1145/2601439

1. INTRODUCTION

This article presents the problem of prominent streak discovery in sequence data. 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, customer transaction sequence, and social media statistics. 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, consecutive days of frequent mentioning of a person in social media, and so on.

It is insightful to investigate prominent streaks because they intuitively and succinctly capture extraordinary subsequences of data. Consider several example application scenarios: (1) business analysts may be interested in prominent streaks in social media usage logs (e.g. streaks of re-tweeting a tweet, streaks of hashtagging a topic); (2) a security auditing may be performed after a streak of excessive login attempts is detected; (3) a cooling system can be started when a streak of days with high temperature has been discovered; and (4) for disease outbreak detection, we can identify 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 [Wong 2004]. The concept of prominent streaks has not yet been studied.

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* [Cohen et al. 2011]. In fact, we witness the mentioning of prominent streaks 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., "LeBron James

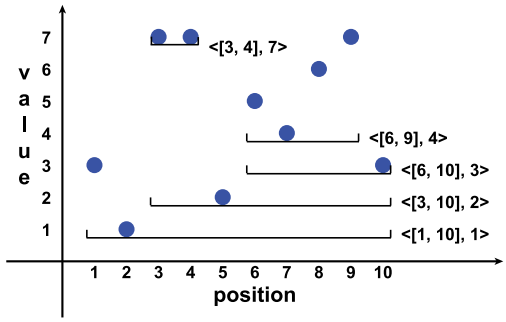


Fig. 1. A data sequence and its prominent streaks.

joined Michael Jordan and Kobe Bryant as the only players," which means that LeBron James's scoring streak mentioned earlier is among the top-3 streaks.)

Given its real-world usefulness and variety, the research on prominent streaks in sequence data opens a spectrum of challenging problems. In an earlier work [Jiang et al. 2011], we proposed the concept of prominent streak and studied the problem of discovering the simplest kind of prominent streaks—that is, those without the aforementioned constraints. In this article, we extend the work to discovering general multidimensional and top-k prominent streaks from multiple sequences, which shall substantially broaden the applicability of our study in real-world scenarios, as evidenced by the stories from news articles presented earlier.

1.1. Problem Definition

Definition 1 (*Streak and Prominent Streak*). Given an n-element sequence $P = (p_1, \ldots, p_n)$, 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 that 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 $v_1 \geq v_2$. For example, $\langle [1, 2], 3 \rangle \prec \langle [4, 7], 6 \rangle$ and $\langle [1, 2], 3 \rangle \prec \langle [3, 4], 5 \rangle$, whereas $\langle [1, 2], 3 \rangle$ and $\langle [7, 8], 3 \rangle$ do not dominate each other.

With regard to $P = (p_1, \ldots, p_n)$, 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 $\mathcal{S}_P$—that is, $\nexists s'$ such that $s' \in \mathcal{S}_P$ 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 is our running example that shows the assists made by an NBA player in 10 consecutive games $P = (3, 1, 7, 7, 2, 5, 4, 6, 7, 3)$. There are five prominent streaks in P– $\langle [1, 10], 1 \rangle$, $\langle [3, 10], 2 \rangle$, $\langle [6, 10], 3 \rangle$, $\langle [6, 9], 4 \rangle$, $\langle [3, 4], 7 \rangle$. Each streak is represented by a horizontal segment, which crosses the minimal value points in the streak and runs from the left end to the right end of the corresponding interval. For instance, $\langle [6, 9], 4 \rangle$ is a prominent streak of minimal value 4, whose interval is from p_6 to p_9. It captures the fact that the NBA player made at least four assists in four consecutive games (game 6 to game 9). The whole data sequence, $\langle [1, 10], 1 \rangle$, 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 nonprominent streaks because it is dominated by $\langle [3, 4], 7 \rangle$.

Definition 1 focuses on the simplest type of prominent streaks. The concept of prominent streak can be extended in several ways. First, we may be interested in top-k prominent streaks that are dominated by less than k other streaks. Second, we may need to compare streaks from not only the same sequence but also multiple different

sequences (e.g., sequences corresponding to different NBA players, cities, stocks). Third, the data points in a sequence can be multidimensional, leading to the pursuit of multidimensional prominent streaks. We have seen examples of all such general prominent streaks at the beginning of Section 1, and their combinations naturally exist. The focus of our following discussion will first be on the simplest prominent streak discovery problem. In Section 5, we discuss how to discover general prominent streaks.

Definition 1 and the problem statement focus on finding streaks of large values. To find streaks of small values (e.g., a stock index below 10,000 for 12 consecutive weeks, described in the aforementioned second news article), two changes should be made. First, a streak should be captured by its interval length and the maximal value (instead of the minimal value) in the interval—that is, $v = \max_{l \le i \le r} p_i$. Second, the dominance relation between streaks should be defined to prefer smaller values. More specifically, s_1 *dominates* s_2 if $r_1 - l_1 \ge r_2 - l_2$ and $v_1 < v_2$ (instead of $v_1 > v_2$), or $r_1 - l_1 > r_2 - l_2$ and $v_1 \le v_2$ (instead of $v_1 \ge v_2$). Given that the new definition would be exactly symmetric to Definition 1, finding streaks of large and small values become the same problem. Hence, we only consider finding streaks of large values in the rest of this article.

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 pairwise streak comparison would be $\binom{|\mathcal{S}_P|}{2} = \frac{n^4+2n^3-n^2-2n}{8}$. Given a sequence of length 10,000, 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 10,000 values. A 1-year usage log for a Web site has 8,760 values at hourly intervals.

Prominent streaks are in fact skyline points [Börzsönyi et al. 2001] 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—that is, 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 nonprominent streaks without exhaustively considering all possible streaks. For *skyline operation*, we apply efficient algorithms from the rich literature on this topic [Börzsönyi et al. 2001; Tan et al. 2001; Kossmann et al. 2002; Papadias et al. 2005]. 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 [Börzsönyi et al. 2001].

Candidate Streak Generation

We considered three methods with increasing pruning power in candidate generation: a baseline method, a nonlinear local prominent streak (NLPS)-based method, and a linear local prominent streak (LLPS)-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. Thus, the baseline method does not have pruning power. 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, that there cannot be a supersequence of s with greater or equal minimal value. In other words, s

ALGORITHM 1: Baseline Method

Input: Data sequence $P = (p_1, \ldots, p_n)$
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 such that 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
```

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. The NLPS-based method finds a superset of LPSs as candidates, whereas the LLPS-based method guarantees to find only LPSs.

Skyline Operation

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.

Our focus is not to compare various skyline algorithms. Many existing algorithms can be adopted. 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 [Börzsönyi et al. 2001]. 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 Börzsönyi et al. [2001]. 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—for instance, streaks with equal lengths and equal minimal values. Hence, the streaks in *skyline* are ordered by v (or by $r - l$). Suppose that the candidate streak is $s = \langle [l', r'], v' \rangle$. We find in *skyline* a pivoting streak $s_i = \langle [l_i, r_i], v_i \rangle$ such that i is the largest index with $v_i \leq v'$—that is, $v_i \leq v' < v_{i+1}$. The following Property 1 says that s must be dominated by s_i or s_{i+1} if it is dominated by any point in *skyline*, and Property 2 says that s can only dominate s_i and its immediate neighbors with smaller v values. (For concise presentation, in these properties, we omit the discussion of boundary cases, i.e., $i = 0$ or $i = |skyline|$.) For quickly finding s_i and its neighbors, we use a balanced binary search tree (BST) on v to store *skyline*. (Thus, we call it the BST-based skyline method.)

PROPERTY 1. *A candidate streak $s = \langle [l', r'], v' \rangle$ is dominated by some points in* skyline *if and only if s is dominated by s_i or s_{i+1}, in which $s_i = \langle [l_i, r_i], v_i \rangle$ and i is the largest index such that $v_i \leq v'$—that is, $v_i \leq v' < v_{i+1}$.*

PROOF. We first prove that if there exists $j < i$ such that $s_j = \langle [l_j, r_j], v_j \rangle \succ s$, then $s_i \succ s$. Since i is the largest index such that $v_i \leq v'$, we have $v_j \leq v_i \leq v'$. Given that $s_j \succ s$, we know $v_j = v_i = v'$ and $r_j - l_j > r' - l'$. From $v_j = v_i$, we know that $r_j - l_j = r_i - l_i$; otherwise, they cannot both exist in *skyline*. Therefore, $s_i \succ s$.

We then prove that if there exists $j > i + 1$ such that $s_j = \langle [l_j, r_j], v_j \rangle \succ s$, then $s_{i+1} \succ s$. Since the points in *skyline* are ordered by v, $v_{i+1} \leq v_j$ and $r_{i+1} - l_{i+1} \geq r_j - l_j$. We already know that $v' < v_{i+1}$ and $r_j - l_j \geq r' - l'$ (since $s_j \succ s$). Therefore, $s_{i+1} \succ s$. □

PROPERTY 2. *If $s = \langle [l', r'], v' \rangle$ dominates totally k streaks in* skyline*, then the k streaks are $s_i, s_{i-1}, \ldots, s_{i-k+1}$.*

PROOF. Since the points in *skyline* are ordered by v, we know that $v_i \leq v_j$ and $r_i - l_i \geq r_j - l_j$ if $i < j$. So, s cannot dominate any s_j such that $j > i$ because $v' < v_{i+1} \leq v_j$. If s dominates s_i, then $v' \geq v_i$ and $r' - l' \geq r_i - l_i$. Since v_i decreases by i and $r_i - l_i$ increases by i, the k streaks dominated by s must be consecutively ordered. □

In comparison with the sorting-based method, the preceding 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 6. 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(BNL)-based method in Börzsönyi et al. [2001] can be applied to fall back on secondary memory. The small size of dynamic skyline also means a small number of streak comparisons. Intuitively, given c candidate streaks, a fast comparison-based sorting algorithm (say quicksort) requires $O(c \log c)$ comparisons, whereas 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 6 show that s is typically much smaller than c.

Monitoring Prominent Streaks

A 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. The aforementioned algorithms naturally fit into such monitoring scenario, with only minor modification. The details are given in Section 4.

1.3. Summary of Contributions and Outline

To summarize, our work makes the following contributions:

—We define 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 except our prior work [Jiang et al. 2011].
—We propose 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 LPS. Both the NLPS-based method and the LLPS-based method 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 extend the solution framework to discovering general prominent streaks. Although the extensions to top-k and multisequence prominent streaks are simple, the extension to multidimensional prominent streak is nontrivial. These extensions significantly broaden the real-world application scenarios of the work.
—We conduct 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 article 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. Section 5 extends the concept of prominent streak and the algorithms for finding general prominent streaks. Experiment setup and results are reported in Section 6. Section 7 concludes the article.

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 [Agrawal et al. 1993; Faloutsos et al. 1993; Agrawal et al. 1995; Yi et al. 1998], finding sequential patterns [Agrawal and Srikant 1995; Srikant and Agrawal 1996; Zaki 2001; Pei et al. 2004; Yan et al. 2003], classification and clustering of sequence and time series data [Smyth 1997; Oates et al. 1999; Liao 2005; Shin and Fussell 2007], biological sequence analysis [Altschul et al. 1990; Rabiner 1989], and so on. However, we are not aware of prior work on the prominent streak discovery problem proposed in this article.

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. The notion of skyline is useful in several applications, including multicriteria decision making. Skyline query has been intensively studied over the past decade. Kung et al. [1975] first proposed in-memory algorithms to tackle the skyline problem, which they called the *maximal vector problem*. Börzsönyi et al. [2001] considered the problem in database context and integrated skyline operator into the database system. They also invented a BNL algorithm and extended the divide-and-conquer algorithm from Kung et al. [1975]. Chomicki et al. [2003] presented the sort-filter-skyline algorithm, which improves upon the BNL algorithm by presorting tuples with a function compatible with the skyline criteria. We apply skyline algorithms over candidate streaks, but our methods are orthogonal to specific choices of skyline algorithms.

A dataset may have too many skyline tuples, especially when the dimensionality of the data is high. Various approaches have been proposed to alleviate this problem. For example, Pei et al. [2006] and Tao et al. [2006] proposed to perform skyline analysis in subspaces instead of the original full space. Several methods were designed to find the representatives among a large number of skyline points [Zhang et al. 2005; Chan et al. 2006; Lin et al. 2007; Tao et al. 2009].

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 [Tan et al. 2001], the nearest neighbor search algorithm [Kossmann et al. 2002], and the branch-and-bound skyline algorithm [Papadias et al. 2005]. Other variants of skyline queries have also been studied, including skyline cube, which aims to answer skyline queries over any combination of dimensions [Pei et al. 2006; Xia and Zhang 2006].

Jiang and Pei [2009] 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 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 values are close to each other (within a given threshold) and are no smaller than the values outside the interval [Wang and Wang 2006]. The plateau problem is not concerned about comparing different intervals.

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 LPS, we propose the NLPS- and LLPS-based methods. Both drastically reduce the number of candidate streaks in practice. LLPS further guarantees only a linear number of candidate streaks.

3.1. Local Prominent Streak

Definition 2 (*Local Prominent Streak*). Given a sequence of data values $P = (p_1, \ldots, p_n)$, we say a streak $s = \langle [l, r], v \rangle \in \mathcal{S}_P$ is an LPS or *locally prominent* if there does not exist any other streak $s' = \langle [l', r'], v' \rangle \in \mathcal{S}_P$ such that $[l', r'] \supset [l, r]$ and $s' \succ s$. (That is, 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 LPSs in sequence P as $\mathcal{LPS}_P$.

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

PROPERTY 3. *Every prominent streak is also an LPS—that is,* $\mathcal{PS}_P \subseteq \mathcal{LPS}_P$.

PROOF. Suppose that there is a prominent streak that is not locally prominent (i.e., $\exists s \in \mathcal{PS}_P$ such that $s \notin \mathcal{LPS}_P$). 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 that s is not dominated by any other streak. Therefore, a streak cannot be prominent if it is not even locally prominent. □

Property 3 is illustrated by Figure 2, as all prominent streaks in Figure 1 also appear in Figure 2. However, the reverse of Property 3 does not hold—LPSs are not necessarily

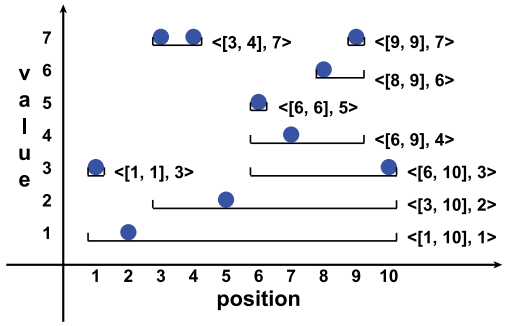


Fig. 2. Local prominent streaks.

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 that* $s = \langle [l, r], v\rangle$ *and* $s' = \langle [l', r'], v'\rangle$ *are two different LPSs in* P*—that is,* $s, s' \in \mathcal{LPS}_P$, $l \neq l'$, *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* $k \neq k'$*—that is,* $argmin_{i\in[l,r]} p_i \cap argmin_{i\in[l',r']} p_i = \emptyset$.

PROOF. If $[l, r] \cap [l', r'] = \emptyset$ (i.e., the two intervals do not overlap), it is obvious that $k \neq k'$. Now consider the case when $[l, r] \cap [l', r'] \neq \emptyset$—that is, $l \leq l' \leq r$ 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 that 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 that $s'' \succ s$ 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 4. $|\mathcal{LPS}_P| \leq |P|$.

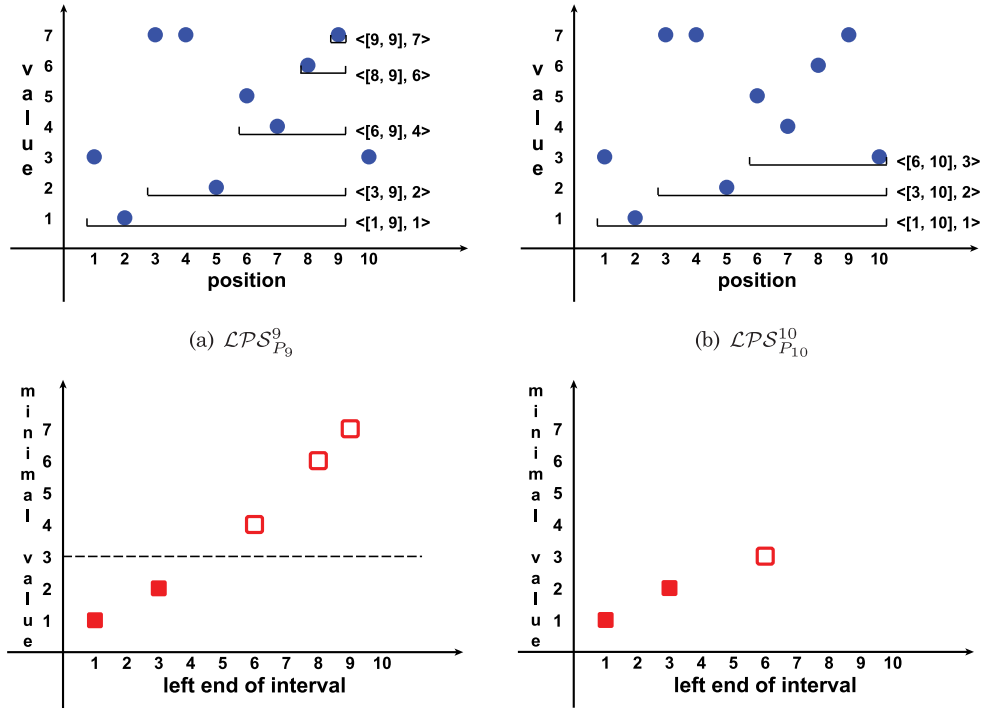
From Property 3, we know that $\mathcal{LPS}_P$ is a sufficient candidate set for $\mathcal{PS}_P$—that is, we can guarantee to find all prominent streaks if we only consider LPSs. Property 4 further shows how small $\mathcal{LPS}_P$ is and thus how good it is as a candidate set. Specifically, the size of $\mathcal{LPS}_P$ is at most $|P|$, the length of the sequence, in contrast to 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 $\mathcal{LPS}_{P_k}^k$

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

Definition 3. $\mathcal{LPS}_P^k$ is the set of LPSs in P that end at position k—that is, $\mathcal{LPS}_P^k = \{s | s \in \mathcal{LPS}_P \text{ and } s = \langle [l, k], v\rangle\}$.

[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 $[l', r']$.



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

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

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

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

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

PROPERTY 5. $\mathcal{LPS}^k_P \subseteq \mathcal{LPS}^k_{P_k}$.

PROOF. Consider any streak $s \in \mathcal{LPS}^k_P$. 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. Thus, $s \in \mathcal{LPS}^k_{P_k}$. □

Consider the running example again. Figure 3(a) shows $\mathcal{LPS}^9_{P_9}$, 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, $\mathcal{LPS}^9_P$ contains

ALGORITHM 3: Nonlinear LPS Method

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^k_{P_k} by Algorithm 4
4 |   for each streak s in LPS^k_{P_k} do
5 |   |   skyline ← skyline_update(skyline, s)
```

$\langle [6, 9], 4\rangle$, $\langle [8, 9], 6\rangle$, $\langle [9, 9], 7\rangle$. Streaks $\langle [1, 9], 1\rangle$ and $\langle [3, 9], 2\rangle$ do not belong to $\mathcal{LPS}_P$ and thus do not belong to $\mathcal{LPS}^9_P$ since they are locally dominated by $\langle [1, 10], 1\rangle$ 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 nine values of P.

3.3. Nonlinear LPS Method

By Property 5 and the fact that $\mathcal{LPS}^1_P, \ldots, \mathcal{LPS}^{|P|}_P$ is a partition of $\mathcal{LPS}_P$, we have

$$\mathcal{LPS}_P = \bigcup_{1\le k\le |P|} \mathcal{LPS}^k_P \subseteq \bigcup_{1\le k\le |P|} \mathcal{LPS}^k_{P_k}. \tag{1}$$

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 6.

Along this line, Algorithm 3 presents the method to compute candidate streaks. Since the number of candidates may be superlinear to the length of data sequence, it is referred to as an 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.

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 of the streaks in LPS^k_{P_k}.
```

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 that consists of extensions of all 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 that $s_2 = \langle [l_1, k-1], v_2\rangle \notin \mathcal{LPS}^{k-1}_{P_{k-1}}$. 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 $v_4 = \min\{v_3, p_k\} \ge \min\{v_2, p_k\} = v_1$. Thus, $s_4 \succ s_1$, which contradicts with the precondition 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.

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,

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

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

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

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

—$\mathcal{LPS}^{k-1\,<}_{P_{k-1}}$: 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 of the new streaks belong to $\mathcal{LPS}^k_{P_k}$. For the running example, corresponding to $\mathcal{LPS}^{9\,<}_{P_9}$, we have $S1 = \{\langle [1, 10], 1\rangle, \langle [3, 10], 2\rangle\}$.

—$\mathcal{LPS}^{k-1\,\ge}_{P_{k-1}}$: We use $S2$ to denote the right-end extension of $\mathcal{LPS}^{k-1\,\ge}_{P_{k-1}}$. Since every streak in $\mathcal{LPS}^{k-1\,\ge}_{P_{k-1}}$ 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 $\mathcal{LPS}^k_{P_k}$. In other words, we only need to extend the longest streak in $\mathcal{LPS}^{k-1\,\ge}_{P_{k-1}}$ to form a new candidate streak. Furthermore, since every streak in $S2$ has the same r value (the right end of the interval)—that is, 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 $\mathcal{LPS}^{9\,\ge}_{P_9}$, we

have $S2 = \{\langle[6, 10], 3\rangle, \langle[8, 10], 3\rangle, \langle[9, 10], 3\rangle\}$. The longest streak in $S2$ is $\langle[6, 10], 3\rangle$. It is clear that $\langle[6, 10], 3\rangle$ dominates other streaks in $S2$. Hence, it belongs to $\mathcal{LPS}^{10}_{P_{10}}$.

—$\mathcal{LPS}^{k-1\,\geq}_{P_{k-1}} = \emptyset$: If $\mathcal{LPS}^{k-1\,\geq}_{P_{k-1}}$ 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 preceding discussion is captured by the following Property 6.

PROPERTY 6. *$\mathcal{LPS}^{k}_{P_k} = S1 \cup \{S2^*\}$ if $S2 \neq \emptyset$ and $\mathcal{LPS}^{k}_{P_k} = S1 \cup \{\langle[k, k], p_k\rangle\}$ if $S2 = \emptyset$.*

We use Figure 3 to explain the procedure previously shown 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 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 five points represent the five 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 10th data entry $p_{10} = 3$. It bisects $\mathcal{LPS}^{9}_{P_9}$ into $\mathcal{LPS}^{9\,\geq}_{P_9}$ (three hollow points above the line) and $\mathcal{LPS}^{9\,<}_{P_9}$ (two 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 two filled points in Figure 3(d), corresponding to $\langle[1, 10], 1\rangle$ and $\langle[3, 10], 2\rangle$. Among the streaks extended from $\mathcal{LPS}^{9\,\geq}_{P_9}$, 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). More specifically, their v and l values both strictly monotonically increase, from the bottom of the stack to the top. The monotonicity on l is obvious, as they are different streaks of the same r value. The monotonicity on v thus is also clear because their length monotonically decreases due to monotonically increasing l, and they must not dominate each other. 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 left-most point denotes the bottom of the stack (with the smallest v), whereas the right-most point denotes the top of the stack (with the largest v). After data entries $p_1, \ldots, p_{k-1}$ are encountered, *lps* contains $\mathcal{LPS}^{k-1}_{P_{k-1}}$. Given data entry p_k, we popped from the stack all streaks whose v values are greater than or equal to p_k. Among the popped streaks, the left-most 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.[2] In each invocation, exactly one item is pushed into the stack.

[2]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 LPS for P_1.

ALGORITHM 5: Linear LPS Method

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 $\mathcal{LPS}_P^{k-1}$ and $\mathcal{LPS}_{P_k}^k$

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

```
// Insert the following line before Line 1 in Algorithm 4.
1 LPS_P^{k-1} ← ∅
// Insert the following two lines after Line 6 in Algorithm 4, in the same else
   branch as Line 6.
2 if pivot.v > p_k then
3 |   LPS_P^{k-1} ← LPS_P^{k-1} ∪ {pivot}
```

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 $\mathcal{LPS}_{P_k}^k$ and include them into candidate streaks. Thus, for an n-element sequence, the total number of candidate streaks considered is $\sum_{k=1}^{n} |\mathcal{LPS}_{P_k}^k|$. 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 $\mathcal{LPS}_{P_1}^1 = \{\langle[1, 1], 10\rangle\}$, $\mathcal{LPS}_{P_2}^2 = \{\langle[1, 2], 10\rangle, \langle[2, 2], 20\rangle\}$, and $\mathcal{LPS}_{P_3}^3 = \{\langle[1, 3], 10\rangle, \langle[2, 3], 20\rangle, \langle[3, 3], 30\rangle\}$.

3.4. Linear LPS Method

Now we present 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 $\mathcal{LPS}_{P_k}^k$ from $\mathcal{LPS}_{P_{k-1}}^{k-1}$ when the k-th data entry is encountered, for k from 1 to n. However, different from Algorithm 3, it also computes $\mathcal{LPS}_P^{k-1}$ from $\mathcal{LPS}_{P_{k-1}}^{k-1}$. Computation of both $\mathcal{LPS}_{P_k}^k$ and $\mathcal{LPS}_P^{k-1}$ is done in Algorithm 6, which is a simple extension of Algorithm 4. It is worth noting that, since $P_n = P$, $\mathcal{LPS}_P^n$ and $\mathcal{LPS}_{P_n}^n$ are identical.

To produce $\mathcal{LPS}_P^{k-1}$ from $\mathcal{LPS}_{P_{k-1}}^{k-1}$ given the k-th entry p_k, Algorithm 6 is based on the following Property 7. Its intuition is as follows. Recall that the minimal value of any streak in $\mathcal{LPS}_{P_{k-1}}^{k-1\geq}$ (Equation (3)) is not smaller than p_k. It follows that if the minimal value of a streak in $\mathcal{LPS}_{P_{k-1}}^{k-1\geq}$ is greater than p_k, the streak cannot grow into a longer LPS without changing the minimal value. Hence, the streak itself is an LPS. To

summarize, $\mathcal{LPS}_P^{k-1}$ is the same as $\mathcal{LPS}_{P_{k-1}}^{k-1\geq}$. The only exception is the longest streak in $\mathcal{LPS}_{P_{k-1}}^{k-1\geq}$—that is, 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 $\mathcal{LPS}_P^{k-1}$, because it can be right extended and included in $\mathcal{LPS}_P^{k'}$ for some $k' \geq k$.

LEMMA 3. *For an n-entry sequence P, a streak $s = \langle [l, r], v\rangle$ is an LPS if and only if $(l = 1$ or $v > p_{l-1})$ and $(r = n$ or $v > p_{r+1})$.*

PROOF. We prove by contradiction. Consider $l > 1$. If $v \leq p_{l-1}$, then s is dominated by $\langle [l-1, r], v\rangle$, which contradicts with s being an LPS. Consider $r < n$. Similarly if $v \leq p_{r+1}$, then s is dominated by $\langle [l, r+1], v\rangle$, which contradicts with s being locally prominent. □

PROPERTY 7. *Given an n-entry sequence P, for any position $1 < k \leq n$, $\mathcal{LPS}_P^{k-1} = \{s | s = \langle [l, k-1], v\rangle \in \mathcal{LPS}_{P_{k-1}}^{k-1\geq}$ and $v > p_k\}$.*

PROOF. Proof of the equality from left to right: suppose that streak $s = \langle [l, k-1], v\rangle \in \mathcal{LPS}_P^{k-1}$. By Property 5 $s \in \mathcal{LPS}_{P_{k-1}}^{k-1}$, and by Lemma 3 $v > p_k$. By the concept of $\mathcal{LPS}_{P_{k-1}}^{k-1\geq}$ in Equation (3), $s \in \mathcal{LPS}_{P_{k-1}}^{k-1\geq}$.

Proof of the equality from right to left: suppose that streak $s = \langle [l, k-1], v\rangle$ satisfies $s \in \mathcal{LPS}_{P_{k-1}}^{k-1\geq}$ and $v > p_k$. Then, s is an LPS in the scope of $\mathcal{LPS}_{P_{k-1}}^{k-1}$, which means, by Lemma 3, that $l = 1$ or $v > p_{l-1}$. Since $v > p_k$, by Lemma 3 s is an LPS in P. Therefore, $s \in \mathcal{LPS}_P^{k-1}$. □

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

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

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 until any particular moment. Formally, given a continuously growing data sequence P (e.g., 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.

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

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

ALGORITHM 7: Continuous Monitoring of Prominent Streaks

```
Input: The new data entry p_k
Compute LPS_P^{k-1} and LPS_{P_k}^k by Algorithm 6
if last_requested_position < k − 1 then
    for each streak s in LPS_P^{k-1} do
        skyline ← sklyine_update(skyline, s)
if PS_{P_k} is requested then
    for each streak s in LPS_{P_k}^k do
        skyline ← sklyine_update(skyline, s)
    last_requested_position ← k
    // Now, skyline contains all prominent streaks in PS_{P_k}
```

the next iteration of the outer loop in Algorithms 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 $\mathcal{LPS}_{P_{k-1}}^{k-1}$ to $\mathcal{LPS}_{P_k}^k$ and updates the skyline with the streaks in $\mathcal{LPS}_{P_k}^k$.

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, $\mathcal{LPS}_P^{k-1}$ and $\mathcal{LPS}_{P_k}^k$ are dynamically computed by Algorithm 6. The skyline is updated with the candidate streaks in $\mathcal{LPS}_P^{k-1}$, only if $\mathcal{PS}_{P_{k-1}}$ was not requested by the user when p_{k-1} was visited. Note that if $\mathcal{PS}_{P_{k-1}}$ were requested, the skyline has already been updated with the streaks in $\mathcal{LPS}_{P_{k-1}}^{k-1}$. Since $\mathcal{LPS}_P^{k-1} \subseteq \mathcal{LPS}_{P_{k-1}}^{k-1}$, we do not need to update the skyline with $\mathcal{LPS}_P^{k-1}$ again. Finally, if the user requests $\mathcal{PS}_{P_k}$, then the skyline has to be updated with $\mathcal{LPS}_{P_k}^k$ since all LPSs (with regard to P_k) ending at p_k must be considered. In Section 6, 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).

5. DISCOVERING GENERAL PROMINENT STREAKS

In this section, we extend the concept of prominent streak and the algorithms introduced in previous sections to general cases. Specifically, we investigate how to discover top-k, multisequence, and multidimensional prominent streaks.

5.1. Top-k Prominent Streaks

Definition 5 (*Top-k Prominent Streak*). With regard to a sequence $P = (p_1, \ldots, p_n)$ and its LPSs $\mathcal{LPS}_P$, a streak $s \in \mathcal{LPS}_P$ is a *top-k prominent streak* if it is not dominated by k or more streaks in $\mathcal{LPS}_P$—that is, $|\{s' | s' \in \mathcal{LPS}_P \text{ and } s' \succ s\}| < k$. The set of all top-$k$ prominent streaks in P is denoted by $\mathcal{KPS}_P$. Note that there can be more than k top-k prominent streaks.

Top-k prominent streaks are those LPSs dominated by less than k other LPSs, by Definition 5. This definition has two implications. First, a top-k prominent streak must be locally prominent. For instance, a streak does not qualify even if it is only dominated by one subsuming streak and $k > 1$. Second, a streak can qualify even if it is dominated by k or more other streaks, as long as less than k of those dominating streaks are LPSs.

Consider a sequence $P = (20, 30, 25, 30, 5, 5, 15, 10, 15, 5)$, corresponding to the points made by a basketball player in all of his games. The streak $\langle[3, 4], 25\rangle$, although only dominated by $\langle[2, 4], 25\rangle$, is a substreak of the latter and hence is not a top-2 prominent streak. The intuitive explanation is that $\langle[3, 4], 25\rangle$ is within the interval of $\langle[2, 4], 25\rangle$, and therefore we do not consider it important. On the other hand, the streak $\langle[7, 9], 10\rangle$ is a top-2 prominent streak. Although it is dominated by three streaks, $\langle[1, 4], 20\rangle$, $\langle[1, 3], 20\rangle$, and $\langle[2, 4], 25\rangle$, the dominating streaks are all from the same period and only one of the three is an LPS.

The candidate streak generation methods discussed in previous sections are applicable in discovering top-k prominent streaks. We only need several small changes on skyline operation. For LLPS, since the candidates produced are guaranteed to be LPSs only, we simply need to maintain a counter for each current skyline point in the dynamic skyline. The counter of a point records the number of its dominators in the skyline. When a candidate is compared against current skyline points, it is inserted into the skyline if it has less than k dominators. A current skyline point is removed if its counter reaches k. With regard to the baseline method and NLPS, they may produce candidates that are not LPSs. A candidate must be pruned if another candidate streak dominates it and subsumes it. (Note that they both produce candidates with the same right end of interval at the same time. Therefore, a candidate cannot be locally dominated by existing points in the current skyline.)

5.2. Multisequence Prominent Streaks

Definition 6 (*Multisequence Prominent Streak*). Given multiple sequences $\mathcal{P} = \{P^1, \ldots, P^m\}$ and their corresponding sets of streaks $\mathcal{S}_{P^1}, \ldots, \mathcal{S}_{P^m}$, a streak $s \in \mathcal{S}_{P^i}$ is a *multisequence prominent streak* in $\mathcal{P}$ if there does not exist a streak in any sequence that dominates s. More formally, $\nexists s', j$ such that $s' \in \mathcal{S}_{P^j}$, and $s' \succ s$. The set of all multisequence prominent streaks with regard to $\mathcal{P}$ is $\mathcal{PS}_{\mathcal{P}}$.

As an example, consider three sequences corresponding to the points made by three basketball players in all of their games—$P_1 = (20, 30, 25, 30, 5, 5, 15, 10, 15, 5)$, $P_2 = (10, 5, 30, 35, 21, 25, 5, 15, 5, 25)$, and $P_3 = (5, 10, 15, 5, 25, 10, 20, 5, 15, 10)$. The streak $\langle[1, 4], 20\rangle$ of P_1 is a prominent streak within P_1 itself but is dominated by $\langle[3, 6], 21\rangle$ in P_2. Hence, it is not a multisequence prominent streak.

The extension from single-sequence algorithms (baseline, NLPS, LLPS) to multisequence algorithms is simple. We process individual sequences separately by the single-sequence algorithms and use a common dynamic skyline to maintain their prominent streaks. That is, when an LPS within a sequence P_i is identified, it is compared with current streaks in the dynamic skyline, which contains prominent streaks from all sequences.

5.3. Multidimensional Prominent Streaks

Definition 7 (*Multidimensional Prominent Streak*). In an n-entry d-dimensional sequence $P = (\vec{p}_1, \ldots, \vec{p}_n)$, a point $\vec{p}_i$ is a d-dimensional vector of data values. A streak s in P is an interval-vector pair $\langle[l, r], \vec{v}\rangle$, where

$$\vec{v} = \left(\min_{l \le i \le r} \vec{p}_i[1], \ldots, \min_{l \le i \le r} \vec{p}_i[d]\right), \tag{4}$$

$\vec{p}_i[j]$ is the j-th dimension of $\vec{p}_i$, and $1 \le l \le r \le n$.

A d-dimensional vector $\vec{v} = (\vec{v}[1], \ldots, \vec{v}[d])$ dominates another vector $\vec{v'} = (\vec{v'}[1], \ldots, \vec{v'}[d])$, denoted by $\vec{v} \succ \vec{v'}$, if and only if $\vec{v}(1) \ge \vec{v'}[1], \ldots, \vec{v}[d] \ge \vec{v'}[d]$ and

ALGORITHM 8: Update Dynamic Skyline for Multidimensional Sequences (*skyline_update*)

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

```
1 dominating ← Find streaks in skyline that dominate s, by a range query on the KD-tree
  over skyline
2 if dominating ≠ ∅ then
3 |   return skyline
4 dominated ← Find streaks in skyline that are dominated by s, by another range query on
  the KD-tree
5 Remove dominated from skyline
6 Insert s into skyline
7 return skyline
```

$\exists j$ such that $\vec{v}[j] > \vec{v'}[j]$. Moreover, we use $\vec{v} \succeq \vec{v'}$ to denote the case when $\vec{v}$ dominates or equals $\vec{v'}$.

A streak $s = \langle [l, r], \vec{v} \rangle$ dominates another streak $s' = \langle [l', r'], \vec{v'} \rangle$, denoted by $s \succ s'$, if and only if $r - l \geq r' - l'$ and $\vec{v} \succ \vec{v'}$, or $r - l > r' - l'$ and $\vec{v} \succeq \vec{v'}$.

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 $\mathcal{S}_P$—that is, $\nexists s'$ such that $s' \in \mathcal{S}_P$ and $s' \succ s$. The set of all multidimensional prominent streaks in P is denoted by $\mathcal{PS}_P$.

For a running example in this section, consider a two-dimensional sequence $P = ((10, 10), (40, 20), (40, 30), (30, 40), (50, 30), (20, 30))$. By the preceding definition, there are eight prominent streaks in P—$\langle [1, 6], (10, 10) \rangle$, $\langle [2, 3], (40, 20) \rangle$, $\langle [2, 5], (30, 20) \rangle$, $\langle [2, 6], (20, 20) \rangle$, $\langle [3, 5], (30, 30) \rangle$, $\langle [3, 6], (20, 30) \rangle$, $\langle [4, 4], (30, 40) \rangle$, $\langle [5, 5], (50, 30) \rangle$. Other streaks are not prominent. For instance, $\langle [2, 4], (30, 20) \rangle$ is dominated by $\langle [3, 5], (30, 30) \rangle$.

In finding prominent streaks from a d-dimensional sequence, skyline operations perform a dominance relationship test on $d + 1$ dimensions—d dimensions for data values and one special dimension for streak length. We maintain a KD-tree [Bentley 1975, 1979] on current skyline points. Given a candidate streak, we use a range query on the KD-tree to efficiently find its dominating points in the current skyline and another ranger query to find its dominated points in the current skyline. Specifically, Algorithm 2 is replaced by Algorithm 8 for multidimensional sequences. We do not further discuss how to answer range queries by multidimensional index structures such as KD-tree as it is well studied.

With regard to candidate streak generation, the brute-force baseline method does not require change, except that *min_value* and its calculation in Algorithm 1 are replaced according to the definition of vector $\vec{v}$ in Equation (4). Our focus in the rest of this section is to extend the concept of LPS and its properties to adapt NLPS and LLPS for multidimensional data sequence. Note that Property 3, Property 5, and Lemma 2 still hold and can be proven in the same way as for single-dimensional sequence. We thus will use the result directly without tediously showing the proof. With the adaptation of NLPS and LLPS for multidimensional sequences, the continuous monitoring approach in Algorithm 7 works in the same way.

Definition 8. For a multidimensional sequence P, a streak $s = \langle [l, r], \vec{v} \rangle \in \mathcal{S}_P$ is an LPS if and only if there does not exist any other streak $s' = \langle [l', r'], \vec{v'} \rangle \in \mathcal{S}_P$, such that $[l', r'] \supset [l, r]$ and $s' \succ s$. (That is, there does not exist such s' that $[l', r'] \supset [l, r]$ and $\vec{v'} \succeq \vec{v}$.) We use $\mathcal{LPS}_P$ to denote the set of all LPSs in P.

For a multidimensional sequence, Property 3 still holds. Hence, every prominent streak in a multidimensional sequence P is also an LPS—that is, $\mathcal{PS}_P \subseteq \mathcal{LPS}_P$—and thus we can still find $\mathcal{LPS}_P$ and use it as the set of candidate streaks. Computing LPSs in a multidimensional sequence is quite similar to that in a single-dimensional sequence. The concepts of $\mathcal{LPS}_P^k$ and $\mathcal{LPS}_{P_k}^k$ remain the same, except that the $\vec{v}$ in each streak $\langle [l, r], \vec{v}\rangle$ is a multidimensional vector instead of a single numeric value. Property 5 also holds. Therefore, the essential ideas of NLPS and LLPS algorithms remain unchanged. NLPS iterates k from 1 to $|P|$, progressively computes $\mathcal{LPS}_{P_k}^k$ from $\mathcal{LPS}_{P_{k-1}}^{k-1}$ when the k-th element $\vec{p}_k$ is visited, and includes $\mathcal{LPS}_{P_k}^k$ into candidate streaks. LLPS does not immediately include all of $\mathcal{LPS}_{P_k}^k$ into candidate streaks. Instead, it waits until seeing $\vec{p}_{k+1}$, then computes $\mathcal{LPS}_P^k$ (in addition to $\mathcal{LPS}_{P_{k+1}}^{k+1}$) from $\mathcal{LPS}_{P_k}^k$, and only includes $\mathcal{LPS}_P^k$ into candidate streaks. Hence, LLPS only considers LPSs ($\mathcal{LPS}_P = \bigcup_{k=1}^n \mathcal{LPS}_P^k$) as candidates, whereas NLPS needs to consider more candidates ($\bigcup_{k=1}^n \mathcal{LPS}_{P_k}^k$), since $\mathcal{LPS}_P^k$ is subsumed by $\mathcal{LPS}_{P_k}^k$ according to Property 5.

5.3.1. Key Ideas. Our following discussion focuses on how to compute $\mathcal{LPS}_{P_k}^k$ and $\mathcal{LPS}_P^{k-1}$ from $\mathcal{LPS}_{P_{k-1}}^{k-1}$, when the k-th element $\vec{p}_k$ arrives. To facilitate the discussion, we partition $\mathcal{LPS}_{P_{k-1}}^{k-1}$ into two disjoint sets $\mathcal{LPS}_{P_{k-1}}^{k-1\preceq}$ and $\mathcal{LPS}_{P_{k-1}}^{k-1\not\preceq}$, as shownnext, which are similar to $\mathcal{LPS}_{P_{k-1}}^{k-1<}$ and $\mathcal{LPS}_{P_{k-1}}^{k-1\geq}$ in Equations (2) and (3). $\mathcal{LPS}_{P_{k-1}}^{k-1\preceq}$ is the set of streaks for which the value at any dimension of the vector $\vec{v}$ is not greater than the corresponding value in $\vec{p}_k$. $\mathcal{LPS}_{P_{k-1}}^{k-1\not\preceq}$ is the set of streaks for which $\vec{v}$ is greater than $\vec{p}_k$ on at least one dimension.

$$\mathcal{LPS}_{P_{k-1}}^{k-1\preceq} = \left\{s | s = \langle [l, k-1], \vec{v}\rangle \in \mathcal{LPS}_{P_{k-1}}^{k-1}, \vec{v} \preceq \vec{p}_k\right\}, \tag{5}$$

$$\mathcal{LPS}_{P_{k-1}}^{k-1\not\preceq} = \left\{s | s = \langle [l, k-1], \vec{v}\rangle \in \mathcal{LPS}_{P_{k-1}}^{k-1}, \exists j \in [1, d] \text{ such that } \vec{v}[j] > \vec{p}_k[j]\right\}. \tag{6}$$

For the running example, $\mathcal{LPS}_{P_5}^5$ is divided into $\mathcal{LPS}_{P_5}^{5\preceq} = \{s_1 = \langle [1, 5], (10, 10)\rangle\}$ and $\mathcal{LPS}_{P_5}^{5\not\preceq} = \{s_2 = \langle [2, 5], (30, 20)\rangle, s_3 = \langle [3, 5], (30, 30)\rangle, s_4 = \langle [5, 5], (50, 30)\rangle\}$.

—*Compute $\mathcal{LPS}_P^{k-1}$ from $\mathcal{LPS}_{P_{k-1}}^{k-1}$*: We can prove that $\mathcal{LPS}_P^{k-1}$ is equivalent to $\mathcal{LPS}_{P_{k-1}}^{k-1\not\preceq}$, given by the following property.

PROPERTY 8. $\mathcal{LPS}_P^{k-1} = \mathcal{LPS}_{P_{k-1}}^{k-1\not\preceq}$.

PROOF. Since Property 5 still holds, $\mathcal{LPS}_P^{k-1} \subseteq \mathcal{LPS}_{P_{k-1}}^{k-1}$. Furthermore, $\mathcal{LPS}_{P_{k-1}}^{k-1\preceq}$ and $\mathcal{LPS}_{P_{k-1}}^{k-1\not\preceq}$ disjointly partition $\mathcal{LPS}_{P_{k-1}}^{k-1}$—that is, $\mathcal{LPS}_{P_{k-1}}^{k-1} = \mathcal{LPS}_{P_{k-1}}^{k-1\preceq} \cup \mathcal{LPS}_{P_{k-1}}^{k-1\not\preceq}$ and $\mathcal{LPS}_{P_{k-1}}^{k-1\preceq} \cap \mathcal{LPS}_{P_{k-1}}^{k-1\not\preceq} = \emptyset$. Therefore, we only need to prove that (1) none of the streaks in $\mathcal{LPS}_{P_{k-1}}^{k-1\preceq}$ is in $\mathcal{LPS}_P^{k-1}$ and (2) all streaks in $\mathcal{LPS}_{P_{k-1}}^{k-1\not\preceq}$ are in $\mathcal{LPS}_P^{k-1}$:

(1) $\forall s \in \mathcal{LPS}_{P_{k-1}}^{k-1\preceq}$, $s \notin \mathcal{LPS}_P^{k-1}$. Suppose that $s = \langle [l, k-1], \vec{v}\rangle$. Its right-end extension is $s' = \langle [l, k], \vec{v'}\rangle$, where $\vec{v'}[j] = min(\vec{v}[j], \vec{p}_k[j])$ for $j \in [1, d]$. Since $\vec{v} \preceq \vec{p}_k$ (by Equation (5)), it follows that $\vec{v'} = \vec{v}$ and thus $s' \succ s$. Hence, s cannot be an LPS in P.

(2) $\forall s \in \mathcal{LPS}_{P_{k-1}}^{k-1\not\preceq}$, $s \in \mathcal{LPS}_P^{k-1}$. We prove this by contradiction. Suppose that $s = \langle [l, k-1], \vec{v}\rangle$. Assume that $s \notin \mathcal{LPS}_P^{k-1}$—that is, there exists $s' \succ s$ such that $s' = \langle [l', r'], \vec{v'}\rangle$,

$[l', r'] \supset [l, k-1]$, and $\vec{v'} \succeq \vec{v}$. By Equation (6), $\exists j \in [1, d]$ such that $\vec{v}[j] > \vec{p}_k[j]$. Therefore, $r' = k-1$; otherwise, $r' = k$ and $\vec{v'}[j] <= \vec{p}_k[j] < \vec{v}[j]$, which contradicts with $\vec{v'} \succeq \vec{v}$. From $[l', r'] \supset [l, k-1]$ and $r' = k-1$, we get $l' < l$, which, along with $s' \succ s$, contradicts with $s \in \mathcal{LPS}^{k-1}_{P_{k-1}}$. The contradictions prove that $s \in \mathcal{LPS}^{k-1}_{P}$. □

—*Compute* $\mathcal{LPS}^{k}_{P_k}$ *from* $\mathcal{LPS}^{k-1}_{P_{k-1}}$: We note that Lemma 2 still holds under multidimensional sequence, except $\langle [k, k], \vec{p}_k \rangle$, 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 $\mathcal{LPS}^{k-1}_{P_{k-1}}$ and one extra streak $\langle [k, k], \vec{p}_k \rangle$ that may belong to $\mathcal{LPS}^{k}_{P_k}$ as well. Again, we consider the two disjoint partitions of $\mathcal{LPS}^{k-1}_{P_{k-1}}$, $\mathcal{LPS}^{k-1\preceq}_{P_{k-1}}$ and $\mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}}$, respectively.

(1) The right-end extensions of all streaks in $\mathcal{LPS}^{k-1\preceq}_{P_{k-1}}$ belong to $\mathcal{LPS}^{k}_{P_k}$, by the following property.

PROPERTY 9. *$\forall s \in \mathcal{LPS}^{k-1\preceq}_{P_{k-1}}$, its right-end extension $s' \in \mathcal{LPS}^{k}_{P_k}$.*

PROOF. We prove by contradiction. Suppose that $s = \langle [l, k-1], \vec{v} \rangle$. Its right-end extension is $s' = \langle [l, k], \vec{v'} \rangle$, where $\vec{v'}[j] = min(\vec{v}[j], \vec{p}_k[j])$ for $j \in [1, d]$. Since $s \in \mathcal{LPS}^{k-1\preceq}_{P_{k-1}}$, $\vec{v} \preceq \vec{p}_k$. Therefore, $\vec{v'} = \vec{v}$. If $s' \notin \mathcal{LPS}^{k}_{P_k}$, then there exists $s'' = \langle [l'', k], \vec{v''} \rangle$ such that $s'' \succ s'$ (i.e., $l'' < l$, and $\vec{v''} \succeq \vec{v'}$). Since s'' and s' have the same right end of interval and $l'' < l$, $\vec{v''} \preceq \vec{v'}$. Therefore, $\vec{v''} = \vec{v'} = \vec{v}$. Consider $s''' = \langle [l'', k-1], \vec{v'''} \rangle$—that is, s'' is the right-end extension of s'''. $\vec{v'''} \succeq \vec{v''}$ by definition of right-end extension. Therefore, $\vec{v'''} \succeq \vec{v}$ and thus $s''' \succ s$ (since $l'' < l$). This contradicts with $s \in \mathcal{LPS}^{k-1}_{P_{k-1}}$. □

(2) Given a streak in $\mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}}$, its right-end extension does not always belong to $\mathcal{LPS}^{k}_{P_k}$.

For a single-dimensional sequence, $\mathcal{LPS}^{k-1}_{P_{k-1}}$ was similarly partitioned into $\mathcal{LPS}^{k-1<}_{P_{k-1}}$ and $\mathcal{LPS}^{k-1\geq}_{P_{k-1}}$. Among the streaks in $\mathcal{LPS}^{k-1\geq}_{P_{k-1}}$, the right-end extension of the longest streak belongs to $\mathcal{LPS}^{k}_{P_k}$. If $\mathcal{LPS}^{k-1\geq}_{P_{k-1}}$ is empty, then $\langle [k, k], p_k \rangle$ belongs to $\mathcal{LPS}^{k}_{P_k}$.

For a multidimensional sequence, multiple but not necessarily all streaks in $\mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}}$ can be right extended to streaks in $\mathcal{LPS}^{k}_{P_k}$. This can be simply proven by using the running example. Recall that $\mathcal{LPS}^{5}_{P_5}{}^{\preceq} = \{s_1 = \langle [1, 5], (10, 10) \rangle\}$ and $\mathcal{LPS}^{5}_{P_5}{}^{\not\preceq} = \{s_2 = \langle [2, 5], (30, 20) \rangle, s_3 = \langle [3, 5], (30, 30) \rangle, s_4 = \langle [5, 5], (50, 30) \rangle\}$. Since $\vec{p}_6 = (20, 30)$, the right-end extensions of s_2, s_3, and s_4 are $s'_2 = \langle [2, 6], (20, 20) \rangle$, $s'_3 = \langle [3, 6], (20, 30) \rangle$, and $s'_4 = \langle [5, 6], (20, 30) \rangle$, respectively. It is clear that $s'_2, s'_3 \in \mathcal{LPS}^{6}_{P_6}$ and $s'_4 \notin \mathcal{LPS}^{6}_{P_6}$ since $s'_3 \succ s'_4$.

5.3.2. Efficient Computation. Based on the discussion in Section 5.3.1, in computing $\mathcal{LPS}^{k}_{P_k}$ and $\mathcal{LPS}^{k-1}_{P}$ from $\mathcal{LPS}^{k-1}_{P_{k-1}}$, the key is to partition $\mathcal{LPS}^{k-1}_{P_{k-1}}$ into $\mathcal{LPS}^{k-1\preceq}_{P_{k-1}}$ (which equals $\mathcal{LPS}^{k-1}_{P}$) and $\mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}}$. The right-end extensions of all streaks in $\mathcal{LPS}^{k-1\preceq}_{P_{k-1}}$ belong to $\mathcal{LPS}^{k}_{P_k}$, and all remaining streaks in $\mathcal{LPS}^{k}_{P_k}$ are formed by right-end extensions of streaks in $\mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}}$. Next, we discuss an efficient method of partitioning $\mathcal{LPS}^{k-1}_{P_{k-1}}$ and identifying streaks in $\mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}}$ that should be extended to streaks in $\mathcal{LPS}^{k}_{P_k}$.

—*Partition* $\mathcal{LPS}^{k-1}_{P_{k-1}}$ *into* $\mathcal{LPS}^{k-1\preceq}_{P_{k-1}}$ *and* $\mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}}$: Suppose that there are m streaks in $\mathcal{LPS}^{k-1}_{P_{k-1}}$, which are $s_1 = \langle [l_1, k-1], \vec{v}_1\rangle, \ldots, s_m = \langle [l_m, k-1], \vec{v}_m\rangle$, where $l_1 < \cdots < l_m$. We can prove that there exists t such that $\mathcal{LPS}^{k-1\preceq}_{P_{k-1}} = \{s_1, \ldots, s_t\}$ and $\mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}} = \{s_{t+1}, \ldots, s_m\}$. (Two special cases are $\mathcal{LPS}^{k-1\preceq}_{P_{k-1}} = \emptyset$ (i.e., $t = 0$) and $\mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}} = \emptyset$ (i.e., $t = m$).) The proof is sketched as follows. Since $l_1 < \cdots < l_m$, for any dimension j, the value $\vec{v}_i[j]$ monotonically increases by i (not necessarily strictly increasing)—that is, $\vec{v}_1[j] \le \vec{v}_2[j] \le \cdots \le \vec{v}_m[j]$. It follows that $\vec{v}_1 \prec \vec{v}_2 \prec \cdots \prec \vec{v}_m$. (Note that $\vec{v}_i \ne \vec{v}_{i+1}$ for any i; otherwise, s_i would dominate s_{i+1}, which contradicts with the notion that they all are LPSs in P_{k-1}.) Given $s_{i_1} \in \mathcal{LPS}^{k-1\preceq}_{P_{k-1}}$ and $s_{i_2} \in \mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}}$, it must be that $i_1 < i_2$, otherwise $i_1 > i_2$, $\vec{v}_{i_1} \succ \vec{v}_{i_2}$ and thus $\forall j, \vec{v}_{i_1}[j] \ge \vec{v}_{i_2}[j]$, which contradicts with Equations (5) and (6).

—*Identify streaks in* $\mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}}$ *that should be extended to streaks in* $\mathcal{LPS}^{k}_{P_{k}}$: To find all of those right-end extensions of streaks in $\mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}}$ that belong to $\mathcal{LPS}^{k}_{P_{k}}$, consider the aforementioned partitioning of $\mathcal{LPS}^{k-1}_{P_{k-1}}$ into $\mathcal{LPS}^{k-1\preceq}_{P_{k-1}} = \{s_1, \ldots, s_t\}$ and $\mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}} = \{s_{t+1}, \ldots, s_m\}$, where the m streaks $s_m, \ldots, s_1$ are decreasingly ordered by the left ends of their intervals. For each $s_i = \langle [l_i, k-1], \vec{v}_i\rangle \in \mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}}$, its right-end extension is $s'_i = \langle [l_i, k], \vec{v}'_i\rangle$. The following important property tells us that if $s'_i \not\prec s'_{i-1}$, then s'_i belongs to $\mathcal{LPS}^{k}_{P_{k}}$.

PROPERTY 10. *For each streak* $s_i = \langle [l_i, k-1], \vec{v}_i\rangle \in \mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}}$*, its right-end extension is* $s'_i = \langle [l_i, k], \vec{v}'_i\rangle$. $s'_i \in \mathcal{LPS}^{k}_{P_{k}}$ *if and only if* $s'_i \not\prec s'_{i-1}$.

PROOF. It is apparent that if $s'_i \prec s'_{i-1}$, then $s'_i \notin \mathcal{LPS}^{k}_{P_{k}}$. Thus, our focus is to prove $s'_i \in \mathcal{LPS}^{k}_{P_{k}}$ if $s'_i \not\prec s'_{i-1}$, by contradiction. Assume that $s'_i \not\prec s'_{i-1}$ but $s'_i \notin \mathcal{LPS}^{k}_{P_{k}}$. Hence, $\exists j < i-1$ and $s'_j \succ s'_i$ (and thus $\vec{v}'_j \succeq \vec{v}'_i$). Since $l_j < l_i$, $\vec{v}'_j \preceq \cdots \preceq \vec{v}'_{i-1} \preceq \vec{v}'_i$. Therefore, $\vec{v}'_j = \cdots = \vec{v}'_{i-1} = \vec{v}'_i$. Hence, $s'_{i-1} \succ s'_i$, which contradicts with $s'_i \not\prec s'_{i-1}$. □

Based on the properties discussed in Sections 5.3.1 and 5.3.2 so far, we design an efficient method to compute $\mathcal{LPS}^{k}_{P_k}$ and $\mathcal{LPS}^{k-1}_{P}$ from $\mathcal{LPS}^{k-1}_{P_{k-1}}$. The current skyline points (prominent streaks) after the $(k-1)$-th element is encountered are stored in the aforementioned KD-tree index structure. The streaks in $\mathcal{LPS}^{k-1}_{P_{k-1}}$, $s_m, \ldots, s_1$, are stored in memory by the decreasing order of the left ends of their intervals. Since they have the same right ends of intervals, only the left ends and the corresponding vectors are stored. When the k-th element $\vec{p}_k$ arrives, this method considers the streaks s_i and their right-end extensions s'_i, starting from $i = m+1$ and iteratively decreasing i by 1. (For $i = m+1$, the special streak in consideration is $s'_{m+1} = \langle [k, k], \vec{p}_k\rangle$.) According to Property 10, the method only requires comparing s'_i with its predecessor s'_{i-1}. If $s'_i \prec s'_{i-1}$, then s_i is removed from the memory. Otherwise, s'_i belongs to $\mathcal{LPS}^{k}_{P_k}$ and thus s_i is updated to s'_i in memory. More specifically, the vector $\vec{v}_i$ of s_i needs to be updated to $\vec{v}'_i$, by $\vec{v}'_i[j] = min(\vec{v}_i[j], \vec{p}_k[j])$ for $j \in [1, d]$. The method goes on until $i = t$ such that $\vec{v}_t \preceq \vec{p}_k$. At that moment, the method will take the following actions:

—The streaks scanned so far $(s_m, \ldots, s_{t+1})$ form $\mathcal{LPS}^{k-1\not\preceq}_{P_{k-1}}$, which is equivalent to $\mathcal{LPS}^{k-1}_{P}$. All remaining streaks in $\mathcal{LPS}^{k-1}_{P_{k-1}}$ $(s_t, \ldots, s_1)$ form $\mathcal{LPS}^{k-1\preceq}_{P_{k-1}}$.

ALGORITHM 9: Progressive Computation of $\mathcal{LPS}^k_{P_k}$ on Multidimensional Sequences

Input: $\mathcal{LPS}^{k-1}_{P_{k-1}}$ and $\vec{p}_k$
Output: $\mathcal{LPS}^k_{P_k}$
// When it starts, stack *lps* consists of streaks in $\mathcal{LPS}^{k-1}_{P_{k-1}}$.
1 *temp_stack* ← an empty stack
2 **while** ! *lps.isempty*() **do**
3 **if** *lps.top*().$\vec{v} \preceq \vec{p}_k$ **then**
4 **break**
5 **else**
6 $s = \langle [l_s, k-1], \vec{v}_s \rangle \leftarrow lps.pop()$
7 $s' \leftarrow \langle [l_s, k], \vec{v'}_s \rangle$, where $\vec{v'}_s = (min(\vec{v}_s[1], \vec{p}_k[1]), \ldots, min(\vec{v}_s[d], \vec{p}_k[d]))$
//right-end extension of s
8 **if** *lps.isempty*() **then**
9 *temp_stack.push*(s')
10 **else**
11 $q = \langle [l_q, k-1], \vec{v}_q \rangle \leftarrow lps.top()$
12 $q' \leftarrow \langle [l_q, k], \vec{v'}_q \rangle$, where $\vec{v'}_q = (min(\vec{v}_q[1], \vec{p}_k[1]), \ldots, min(\vec{v}_q[d], \vec{p}_k[d]))$
//right-end extension of q
13 **if** $q' \not\succ s'$ **then**
14 *temp_stack.push*(s')
15 **while** ! *temp_stack.isempty*() **do**
16 *lps.push*(*temp_stack.pop*())
17 **if** *lps.isempty*() *or* *lps.top*().$\vec{v} \not\succeq \vec{p}_k$ **then**
18 *lps.push*($\langle [k, k], \vec{p}_k \rangle$)
// Now, *lps* contains all the streaks in $\mathcal{LPS}^k_{P_k}$.

ALGORITHM 10: Computing $\mathcal{LPS}^{k-1}_P$ and $\mathcal{LPS}^k_{P_k}$ on Multidimensional Sequences

Input: $\mathcal{LPS}^{k-1}_{P_{k-1}}$ and $\vec{p}_k$
Output: $\mathcal{LPS}^{k-1}_P$ and $\mathcal{LPS}^k_{P_k}$

// Insert the following line before Line 1 in Algorithm 9.
1 $\mathcal{LPS}^{k-1}_P \leftarrow \emptyset$
// Insert the following line after Line 6 in Algorithm 9.
2 $\mathcal{LPS}^{k-1}_P \leftarrow \mathcal{LPS}^{k-1}_P \cup \{s\}$

—The streaks in $\mathcal{LPS}^{k-1}_P$ are candidate prominent streaks. They are compared with current skyline points by the aforementioned range queries over the KD-tree on the skyline points. Nondominated candidates are inserted into the KD-tree.
—For all remaining streaks in the memory (i.e., $\mathcal{LPS}^{k-1\ \preceq}_{P_{k-1}}$), their right-end extensions belong to $\mathcal{LPS}^k_{P_k}$. Since their vectors are all dominated by or equivalent to $\vec{p}_k$, their corresponding vectors do not need to be updated. At this moment, all streaks of $\mathcal{LPS}^k_{P_k}$ are stored in memory by the decreasing order of the left ends of their intervals.

More concretely, Algorithms 3 and 5 remain unchanged, and Algorithms 4 and 6 are replaced by Algorithms 9 and 10, respectively.

5.3.3. A Note on "Curse of Dimensionality". For a single-dimensional sequence with n elements, LLPS produces at most n candidates (i.e., LPSs), according to Property 4. This upper bound guarantees LLPS to be an efficient linear-time algorithm. However, the

same property does not hold for multidimensional sequences. Consider an extreme case that is a two-dimensional n-element sequence $(\vec{p}_1, \ldots, \vec{p}_n)$, where $\vec{p}_i = (i, n-i)$. It is not hard to prove that all $\frac{n(n+1)}{2}$ possible streaks in this sequence are prominent streaks and thus automatically are LPSs. This represents the worst case, in which nothing beats the brute-force baseline method.

Although the worst case indicates the rather notorious "curse of dimensionality," our empirical results on multiple datasets are much more encouraging. The results show that the number of prominent streaks and the execution time of LLPS do not increase exponentially by the dimensionality of data. This is mainly because data values fluctuate and are correlated. We investigate these results in more detail in Section 6.

6. EXPERIMENTS

We report and analyze experimental results in this section. The algorithms were implemented in Java. The experiments were conducted on a server with four 2.00GHz Intel Xeon E5335 CPUs running Ubuntu Linux. The limit on the heap size of Java Virtual Machine was set at 512MB. We discuss the results on basic and general prominent streak discovery in Section 6.1 and Section 6.2, respectively.

6.1. Experimental Results on Basic Prominent Streak Discovery

We used multiple real-world datasets, including time series data library,[3] Wikipedia traffic statistics dataset,[4] NYSE exchange data,[5] AOL search engine log,[6] and FIFA World Cup 98 Web site access log.[7] These datasets cover a variety of application scenarios, including meteorology, hydrology, finance, Web log, and network traffic. Table I shows the information of 12 data sequences from these datasets 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.

Examples of Interesting Prominent Streaks Discovered

From 1985 to 1989, there had been more than 1,000 consecutive trading days with morning gold price greater than $300. During this period, there had been a streak of 400 days with a price of more than $400, although the $500 price only lasted 2 days at most.

In Melbourne, Australia, during the years between 1981 and 1990, the weather had been pleasant. There had been more than 2,000 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 6 days. It was in the summer of the year 1981.

More than half of the prominent streaks that we found in the traffic data of the Lady Gaga Wikipedia page were around September 12, 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 2,000 people in every hour for almost 4 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 that they produce candidates and thus the numbers of produced candidates. Table II shows the total

[3]http://robjhyndman.com/TSDL/.
[4]http://dammit.lt/wikistats/.
[5]http://www.infochimps.com/datasets/nyse-daily-1970-2010-open-close-high-low-and-volume.
[6]http://gregsadetsky.com/aol-data/.
[7]http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

Table I. Data Sequences Used in Experiments on Basic Prominent Streak Discovery

Name	Length	Prominent Streaks (#)	Description
Gold	1,074	137	Daily morning gold price in US dollars, 01/1985–03/1989
River	1,400	93	Mean daily flow of Saugeen River near Port Elgin, 01/1988–12/1991
Melb1	3,650	55	The daily minimum temperature of Melbourne, Australia, 1981–1990
Melb2	3,650	58	The daily maximum temperature of Melbourne, Australia, 1981–1990
Wiki1	4,896	58	Hourly traffic to http://en.wikipedia.org/wiki/Main_page, 04/2010–10/2010
Wiki2	4,896	51	Hourly traffic to http://en.wikipedia.org/wiki/Lady_gaga, 04/2010–10/2010
Wiki3	4,896	118	Hourly traffic to http://en.wikipedia.org/wiki/Inception_(film), 04/2010–10/2010
SP500	10,136	497	S&P 500 index, 06/1960–06/2000
HPQ	12,109	232	Closing price of HPQ in NYSE for every trading day, 01/1962–02/2010
IBM	12,109	198	Closing price of IBM in NYSE for every trading day, 01/1962–02/2010
AOL	132,480	127	Number of queries to AOL search engine in every minute over 3 months
WC98	7,603,201	286	Number of requests to World Cup 98 Web site in every second, 04/1998–07/1998

Table II. Number of Candidate Streaks, Basic Prominent Streak Discovery

Name	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

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—for example, $\binom{7603202}{2} = 2.89 \times 10^{13}$ for WC98. By contrast, NLPS only needs to 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 reduces 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 4), which is verified by Table II.

Table III. Execution Time (in Milliseconds), Basic Prominent Streak Discovery

Name	Baseline	NLPS	LLPS
Gold	183	122	13
River	126	84	19
Melb1	385	101	36
Melb2	384	101	35
Wiki1	670	105	46
Wiki2	646	97	46
Wiki3	632	126	48
SP500	4,453	789	116
HPQ	6,285	338	101
IBM	4,228	377	135
AOL	290,744	752	201
WC98	>1 hour	38,999	3,012

Execution Time

The number of candidate streaks directly determines the efficiency of our algorithms. In Table III, 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 III, 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.

When reporting the execution time of these algorithms, 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 1s to load, whereas the loading time of all other datasets was below 30ms.

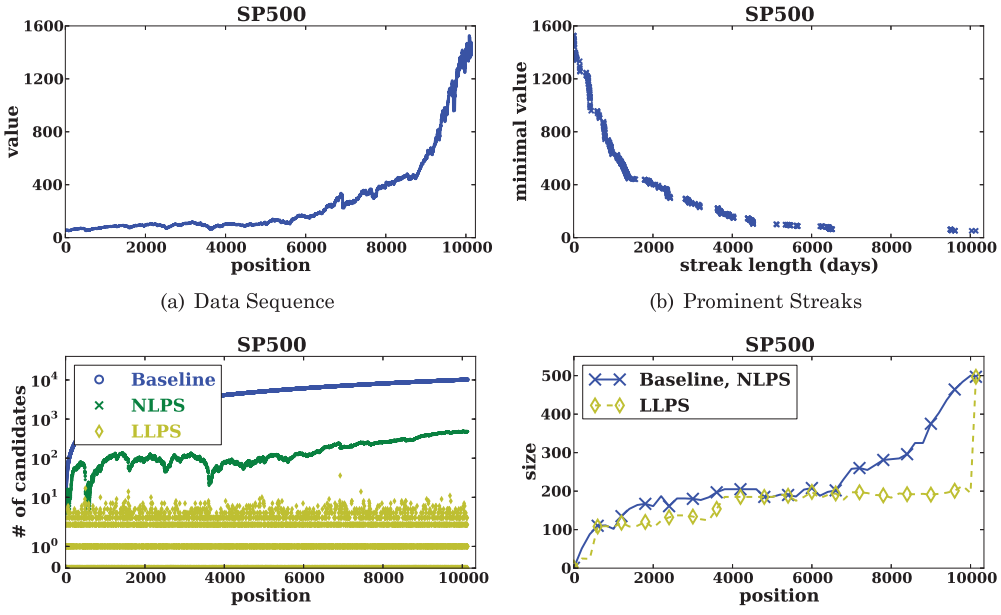
In Table III, we use ">1 hour" to denote the execution time when an algorithm could not finish within 1 hour (i.e., 3,600,000ms). This lower bound is sufficient in showing the performance difference of the various algorithms.

With regard to the comparison of Baseline, NLPS, and LLPS, it is clear from Table III that LLPS outperforms NLPS and that 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 II), 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. 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 I) is the most among all of 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.

In Table II, 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



(a) Data Sequence (b) Prominent Streaks

(c) Number of Candidate Streaks (d) Size of Dynamic Skyline

Fig. 4. Detailed results on SP500, basic prominent streak discovery.

clearly shows the superiority of LLPS, as it always generates orders of magnitude less candidates at each position.

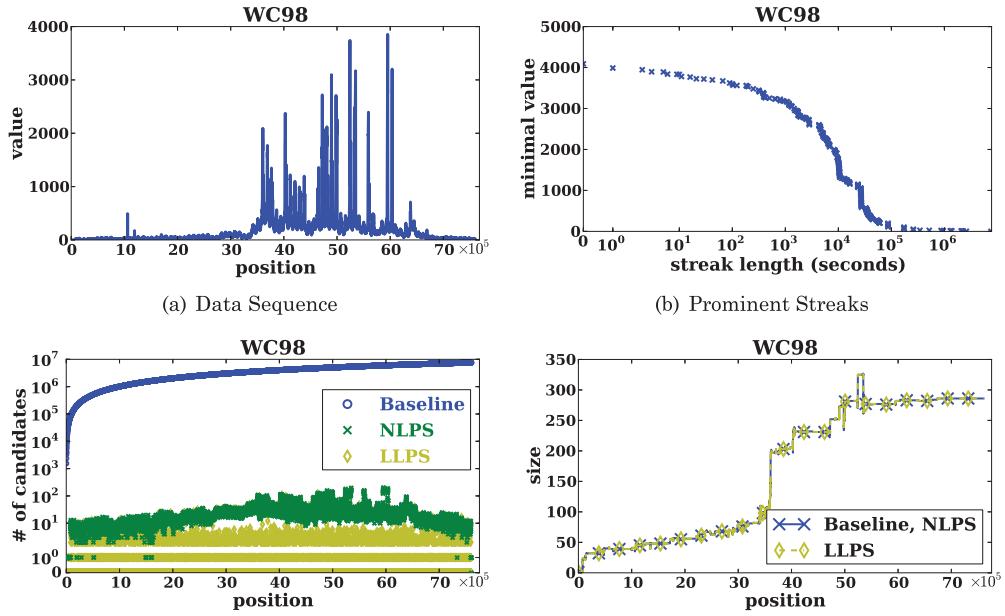
The BST-based skyline method maintains a dynamic skyline, as a BST, in memory. The size of this tree affects the efficiency of tree operations, such as inserting and deleting a streak. Figure 4(d) shows the size of the dynamic skyline along the sequence of SP500 by each algorithm. The curves for Baseline and NLPS overlap since they both store $\mathcal{PS}_{P_k}$, at every position k, in the dynamic skyline. On the 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.

In Figure 5, we show the detailed results on WC98 data, which are similar to the results on SP500 but are also different on several aspects. The data sequence fluctuates. Hence, there are less candidate streaks by NLPS and LLPS, which makes the gap between them and Baseline much bigger. For the same reason, the size of the dynamic skyline is almost identical for the three algorithms. Note that Figure 5(b) uses logarithmic scale on the x-axis, because the very long streaks would otherwise make most other streaks cluttered to the left if linear scale is used.

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.

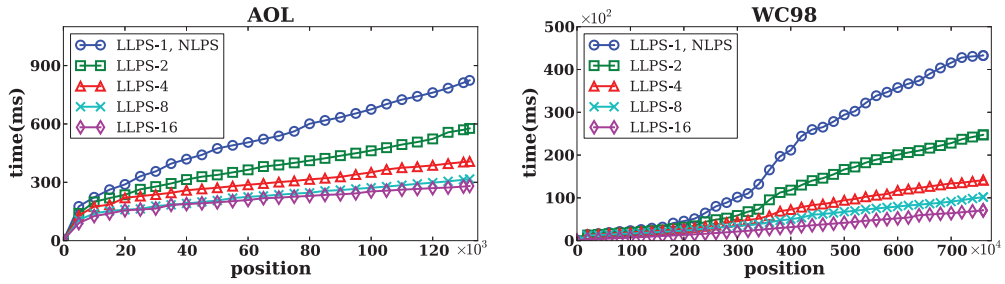
Take AOL and WC98 as examples. Figure 6 shows the execution time of Algorithm 7. The x-axis represents the sequence position, and the y-axis is for the total execution time by that position. There are five curves in each figure, corresponding to five different



(a) Data Sequence (b) Prominent Streaks

(c) Number of Candidate Streaks (d) Size of Dynamic Skyline

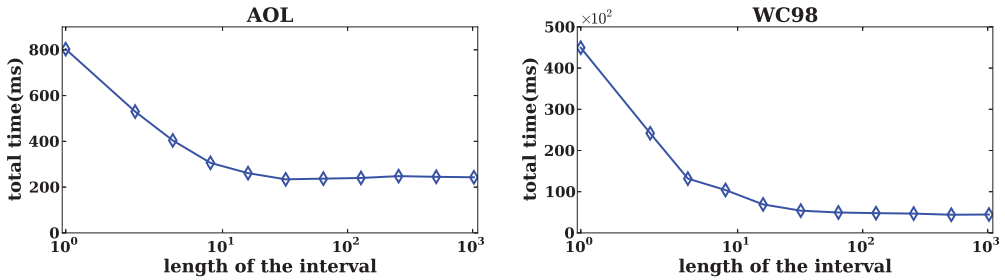
Fig. 5. Detailed results on WC98, basic prominent streak discovery.



(a) AOL (b) WC98

Fig. 6. Cumulative execution time at various positions for different reporting frequencies, basic prominent streak discovery.

frequencies of reporting prominent streaks. For instance, LLPS-1 means that whenever a new data entry comes, all of the prominent streaks so far are reported; LLPS-16 means that 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 6(a) and 6(b) clearly show that the total execution time of LLPS-i increases as the reporting frequency increases (i.e., reporting interval i decreases). Figures 7(a) and 7(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 fairly high (e.g., reporting the prominent streaks at every 16 entries.)



(a) AOL (b) WC98

Fig. 7. Total execution time by reporting frequencies, basic prominent streak discovery.

Table IV. Number of Prominent Streaks and Execution Time (in Milliseconds), Top-5 Prominent Streaks

Name	Prominent Streaks (#)	Baseline	NLPS	LLPS
Gold	147	1884	348	44
River	144	6.81×10^4	275	57
Melb1	160	6.06×10^7	572	96
Melb2	160	3.01×10^6	445	150
Wiki1	181	3.68×10^7	1369	140
Wiki2	115	1.88×10^7	565	172
Wiki3	172	1.05×10^6	473	136
SP500	516	7.09×10^6	13,700	270
HPQ	251	>10 hours	3,211	178
IBM	232	>10 hours	5,914	229
AOL	250	>10 hours	26,000	798
WC98	409	>10 hours	>10 hours	13,300

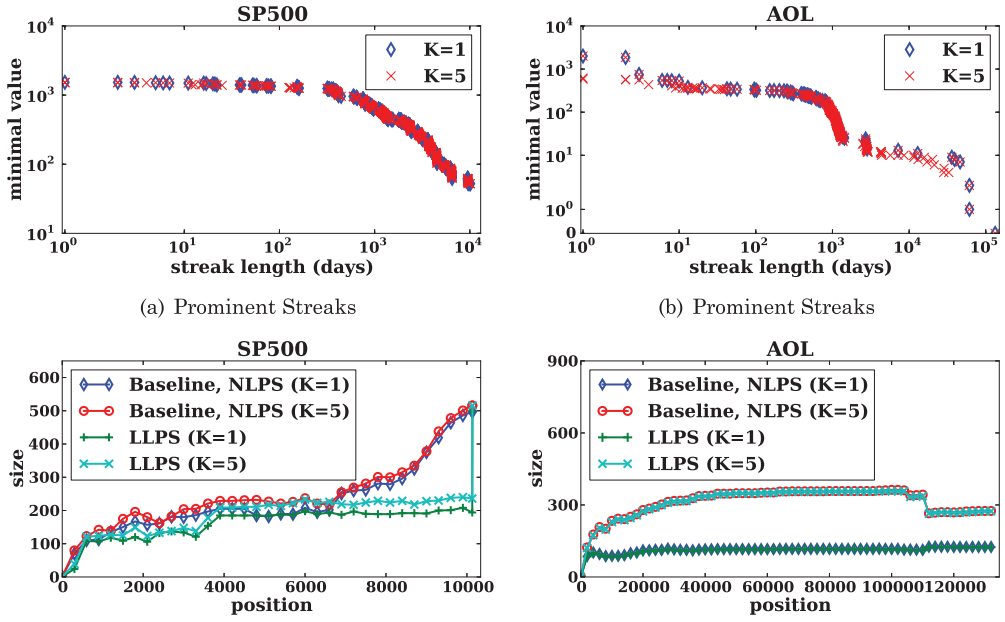
6.2. Experimental Results on General Prominent Streak Discovery

In this section, we discuss the results on top-k, multisequence, and multidimensional prominent streak discovery. At the end of this section, we also present the results from an experiment that put together these different extensions.

Top-k Prominent Streaks

The experiments on top-k prominent streaks were conducted on the same datasets discussed in Section 6.1. For each dataset, Table IV shows the number of top-5 prominent streaks (i.e., $\mathcal{KPS}_P$ in Definition 5) and the execution time of the extended Baseline, NLPS, and LLPS algorithms. Note that the number of candidate streaks shown in Table II remains the same, as the same candidate streak generation methods are used for top-k prominent streaks, as discussed in Section 5.1.

As Table IV shows, in comparison with the execution time in Table III (i.e., the time of discovering top-1 prominent streaks), the execution time of Baseline increased by one or more orders of magnitude, whereas the performance of NLPS and LLPS was degraded by less than one order of magnitude in most cases. This is explained as follows. Finding top-k prominent streaks incurs higher cost of skyline operation than finding top-1 prominent streaks. More specifically, the cost of skyline operation is determined by the number of dominance comparisons between candidate streaks and streaks in the dynamic skyline. Therefore, the number of comparisons increases by both the number of candidate streaks and the size of the dynamic skyline. In comparison



(a) Prominent Streaks (b) Prominent Streaks

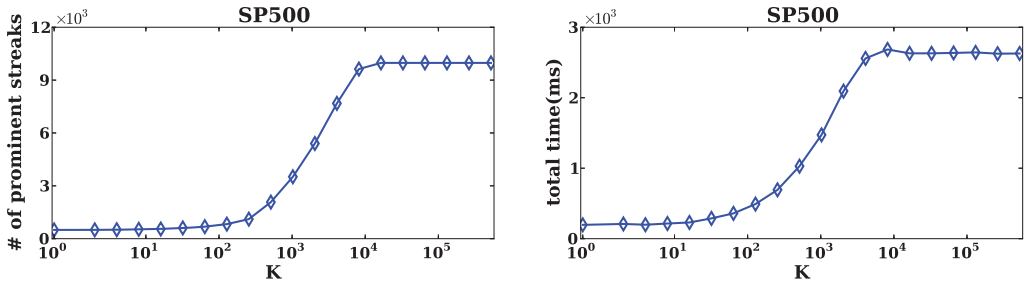
(c) Size of Dynamic Skyline (d) Size of Dynamic Skyline

Fig. 8. Detailed results on SP500 and AOL, top-1 versus top-5 prominent streaks.

with top-1, finding top-k prominent streaks requires maintaining LPSs with as many as $k-1$ dominators, which increases the size of dynamic skyline and thus incurs larger cost. For example, a dominator search for a candidate cannot terminate until the number of dominators reaches k, whereas the search terminates immediately in top-1 algorithms once a dominator is found. The more candidate streaks there are, the larger the increment of skyline operation cost (from top-1 to top-k) grows. This further explains why the performance of Baseline was degraded the most.

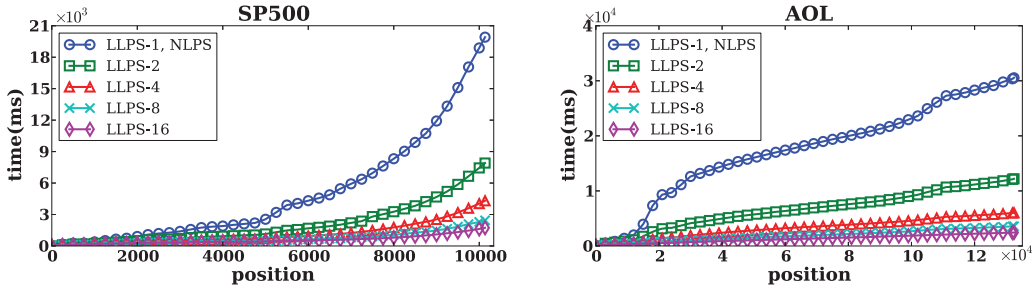
Figure 8 shows some interesting detailed results on two different sequences. Since sequence SP500 increases almost monotonically, an LPS that is not globally prominent most likely has a relatively large number of dominators. Hence, the size of dynamic skyline in top-5 prominent streak discovery is only slightly larger than that in top-1. This explains Figure 8(c). On the contrary, for sequence AOL, the size of dynamic skyline for top-5 is about twice the size of top-1 (Figure 8(d)). This is because sequence AOL fluctuates. The prominent streaks have different right ends of intervals due to the fluctuation. This also explains why the sizes of dynamic skylines in Baseline, NLPS, and LLPS do not differ much from each other in this sequence. However, as Table IV shows, their differences on execution time are still significant because NLPS and LLPS generate much less candidates than Baseline does.

By Definition 5, $\mathcal{PS}_P \subseteq \mathcal{KPS}_P$—that is, all prominent streaks are also top-k prominent streaks. This is clearly shown in Figures 8(a) and 8(b). Furthermore, $\mathcal{KPS}_P \subseteq \mathcal{LPS}_P$—that is, top-$k$ prominent streaks must be LPSs as well. Therefore, the set $\mathcal{KPS}_P$ grows by k and stops growing after k reaches a certain value, when all streaks in $\mathcal{LPS}_P$ are included in $\mathcal{KPS}_P$. This is demonstrated by Figure 9(a), in which the number of prominent streaks in sequence SP500 increases by k until k reaches about 10,000. As a result, total execution time also changes in sync with the number of prominent streaks, as shown in Figure 9(b).



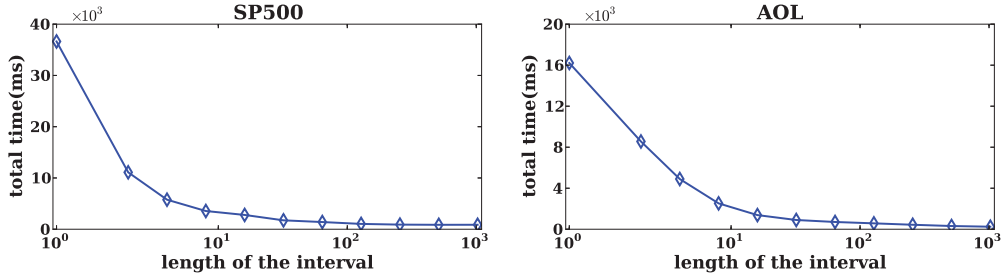
(a) Number of Prominent Streaks (b) Execution Time

Fig. 9. Number of prominent streaks and execution time, LLPS on SP500, top-k prominent streaks, varying k.



(a) SP500 (b) AOL

Fig. 10. Cumulative execution time at various positions for different reporting frequencies, top-5 prominent streak discovery.



(a) SP500 (b) AOL

Fig. 11. Total execution time by reporting frequencies, top-5 prominent streak discovery.

We also experimented with monitoring top-k prominent streaks. The results are shown in Figures 10 and 11, which exhibit patterns of execution time similar to the patterns in Figures 6 and 7 for monitoring top-1 prominent streaks.

Multisequence Prominent Streaks

We used two datasets for experiments on multisequence prominent streak discovery. One (Wiki) is the hourly traffic to Ivy League universities' Wikipedia page[4], one sequence per university. The other dataset (NBA1) contains 1,225 sequences, one sequence per NBA player. Each sequence lists the scores of a player in all of the games

Table V. Data Sequences Used in Experiments on Multisequence Prominent Streak Discovery

Name	Sequences (#)	Average Length	Prominent Streaks (#)	Description
NBA1	1,225	281	28	Points scored by all NBA players from 1991–2004
Wiki	8	14,454	59	Hourly traffic to the Wikipedia pages of Ivy League universities

Table VI. Number of Candidate Streaks, Multisequence Prominent Streak Discovery

Name	Baseline	NLPS	LLPS
NBA1	9.41×10^7	1.23×10^6	3.31×10^5
Wiki	8.36×10^8	1.23×10^6	1.86×10^5

Table VII. Execution Time (in Milliseconds), Multisequence Prominent Streak Discovery

Name	Baseline	NLPS	LLPS
NBA1	3,436	310	292
Wiki	33,537	275	190

that he played from 1991 to 2004.[8] The characteristics of these two datasets are shown in Table V, including the number of sequences, the average sequence length, and the number of prominent streaks. Tables VI and VII show the number of candidate streaks and the execution time, respectively, for Baseline, NLPS, and LLPS. The results are very similar to that in Tables II and III. This is because the process of multisequence prominent streak discovery is not very different from its single-sequence counterpart.

For dataset NBA1, Table VIII shows the distribution of players by the number of prominent streaks contributed by them. All 29 prominent streaks (i.e., NBA scoring records in the period from 1991 to 2004) come from merely 10 different players. Table IX shows the detailed records. One interesting observation from the table is that Karl Malone and John Stockton, two of the healthiest NBA players, had scored in two longest streaks of games. Another example is that Allen Iverson is the only one who scored at least 20 points in more than 50 consecutive games.

Multidimensional Prominent Streaks

We used three datasets for experiments on multidimensional prominent streak discovery, listed in Table X. The first dataset is the game log of NBA player Karl Malone from the 1991 to 2004 seasons.[8] This is a sequence of 986 elements, each of which represents Malone's performance in a game on six performance dimensions. The second dataset is the 2003–2011 Texas motor vehicle crash statistics,[9] a five-dimensional sequence of 3,287 elements, where each element is for 1 day and represents the daily counts of crashes, injuries, fatalities, and so on. The last dataset is the historical NASDAQ stock data of Apple Inc. from 1970 to 2010.[10] In this 6,411-element sequence, each element is for a trading day and contains the opening price, change ratio, and trading volume of the stock of Apple Inc. on that day.

The number of candidate streaks and the execution time by Baseline, NLPS, and LLPS are shown in Tables XI and XII. Figure 12 further shows detailed experimental

[8]http://www.databasebasketball.com/index.htm.
[9]http://www.txdot.gov/government/enforcement/annual-summary.html.
[10]http://www.infochimps.com/datasets/nasdaq-exchange-daily-1970-2010-open-close-high-low-and-volume.

Table VIII. Distribution of Players by Number of Prominent Streaks

Number of Prominent Streaks	Number of Players
0	1,215
1	6
3	1
4	1
5	1
10	1

Table IX. Multisequence Prominent Streaks in Dataset NBA1

Length	Minimal Value	Players
1	71	David Robinson
2	51	Allen Iverson; Antawn Jamison
4	42	Kobe Bryant
9	40	Kobe Bryant
13	35	Kobe Bryant
14	32	Kobe Bryant
16	30	Kobe Bryant
17	27	Michael Jordan
27	26	Allen Iverson
34	24	Tracy McGrady
45	21	Allen Iverson
57	20	Allen Iverson
74	19	Shaquille O'Neal
94	18	Shaquille O'Neal
96	17	Karl Malone
119	16	Karl Malone
149	15	Karl Malone
159	14	Karl Malone
263	13	Karl Malone
357	12	Karl Malone
527	11	Karl Malone
575	10	Karl Malone
758	7	Karl Malone
858	6	Shaquille O'Neal
866	2	Karl Malone
932	1	John Stockton
1,185	0	Jim Jackson

Table X. Data Sequences Used in Experiments on Multidimensional Prominent Streak Discovery

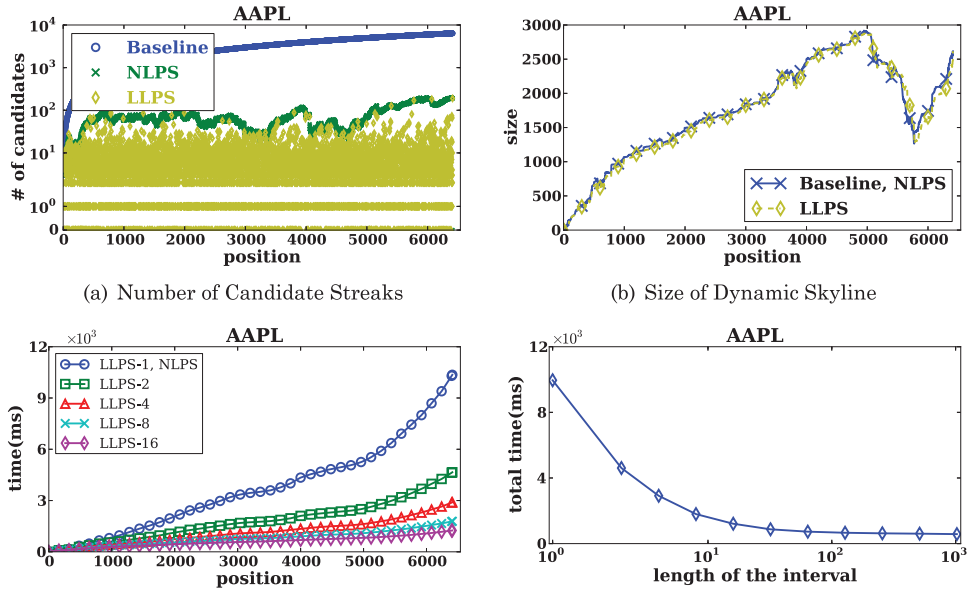
Name	Length	Prominent Streaks (#)	Dimensions (#)	Description
Malone	986	640	6	1991–2004 game log of Karl Malone (minutes, points, rebounds, assists, steals, blocks)
Crashes	3,287	1,493	5	2003–2011 Texas motor vehicle crash statistics (crashes and injuries by date)
AAPL	6,411	2,616	3	NASDAQ stock data of Apple Inc. from 1970 to 2010, on daily values of opening price, change ratio $((open - close)/open \times 100\%)$ and trading volume

Table XI. Number of Candidate Streaks, Multidimensional Prominent Streak Discovery

Name	Baseline	NLPS	LLPS
Malone	4.87×10^5	1.27×10^4	4.47×10^3
Crashes	5.40×10^6	6.95×10^5	1.82×10^4
AAPL	2.06×10^7	4.77×10^5	4.08×10^5

Table XII. Execution Time (in Milliseconds), Multidimensional Prominent Streak Discovery

Name	Baseline	NLPS	LLPS
Malone	4,575	336	180
Crashes	1.08×10^5	1,113	326
AAPL	5.65×10^5	9,997	557



(a) Number of Candidate Streaks

(b) Size of Dynamic Skyline

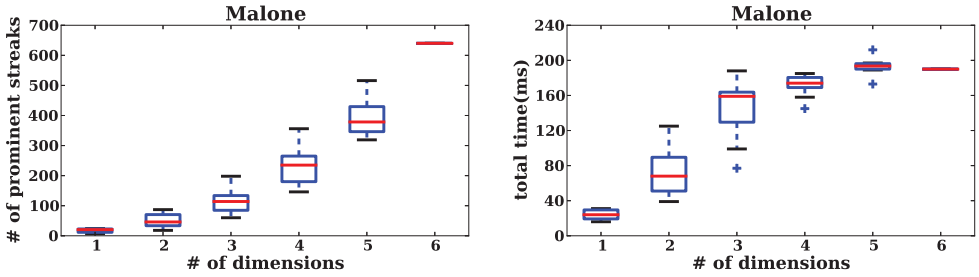
(c) Cumulative Execution Time at Various Positions, for Different Reporting Frequencies

(d) Total Execution Time by Reporting Frequencies

Fig. 12. Detailed results on AAPL, multidimensional prominent streak discovery.

results on dataset AAPL. The observations made on these results are similar to those for basic, top-k, and multisequence prominent streak discovery.

We also investigated how the number of prominent streaks and total execution time of LLPS increase by the dimensionality of data, using dataset Malone. As the boxplots in Figure 13 show, these measures do not increase exponentially by data dimensionality, at least under small dimensionality such as 6. This indicates that while the curse of dimensionality can raise concerns, the empirical results are much more encouraging, partly because data values fluctuate and thus the appearance of a small value terminates many prominent streaks. Furthermore, data values are correlated, which-practically reduces data dimensionality. Finally, the results are for six dimensions at most. We note that arguably the prominent streaks found in the real world, such as the ones in Section 1, mostly would not have more than six dimensions.



(a) Number of Prominent Streaks (b) Execution Time of LLPS

Fig. 13. Experiments on increasing dimensionality.

Table XIII. Data Sequences Used in Experiments on Top-5 Multisequence Multidimensional Prominent Streak Discovery

Name	Sequences (#)	Average Length	Dimensions (#)	Prominent Streaks (#)	Description
NBA2	1,185	290	6	10,867	1991–2004 game log of all NBA players (minutes, points, rebounds, assists, steals, blocks)

Table XIV. Number of Candidate Streaks, Top-5 Multisequence Multidimensional Prominent Streak Discovery

Name	Baseline	NLPS	LLPS
NBA2	9.41×10^7	2.98×10^6	8.76×10^5

Putting It Together: Top-*k* Prominent Streaks on Multiple Multidimensional Sequences

We also used dataset NBA2 (Table XIII) for experiments on discovering top-*k* prominent streaks from multiple multidimensional sequences. This dataset contains 1,185 six-dimensional sequences, each of which corresponds to the game log of an NBA player from 1991 to 2004. One of the sequences is the aforementioned dataset Malone.

Figure 14 shows that distribution of prominent streaks by length. It is clear that the distribution follows the power law, because the minimal value vector for a streak takes the minimal value on each dimension from all elements. The longer a streak is, the smaller the values in its minimal value vector become. Therefore, it is difficult for a long streak to stand out as prominent. Figure 15 shows detailed experimental results on this dataset that show similar patterns to those observed for aforementioned experiments.

7. CONCLUSION AND FUTURE WORK

In this article, 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 LPS. We prove that prominent streaks are a subset of LPSs and that the number of LPSs is less than the length of a data sequence. Our linear LPS-based method guarantees to consider only LPSs, thus achieving significant reduction in candidate streaks. The proposed properties and algorithms are also extended for discovering general top-*k*, multisequence, and multidimensional prominent streaks. The results of experiments over multiple real datasets verified the effectiveness of the proposed methods.

Table XV. Execution Time (in Milliseconds), Top-5 Multisequence Multidimensional Prominent Streak Discovery

Name	Baseline	NLPS	LLPS
NBA2	1.39×10^7	4.33×10^5	1.14×10^5

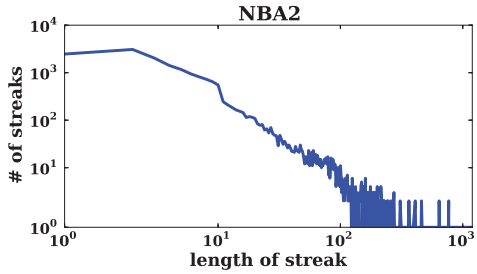
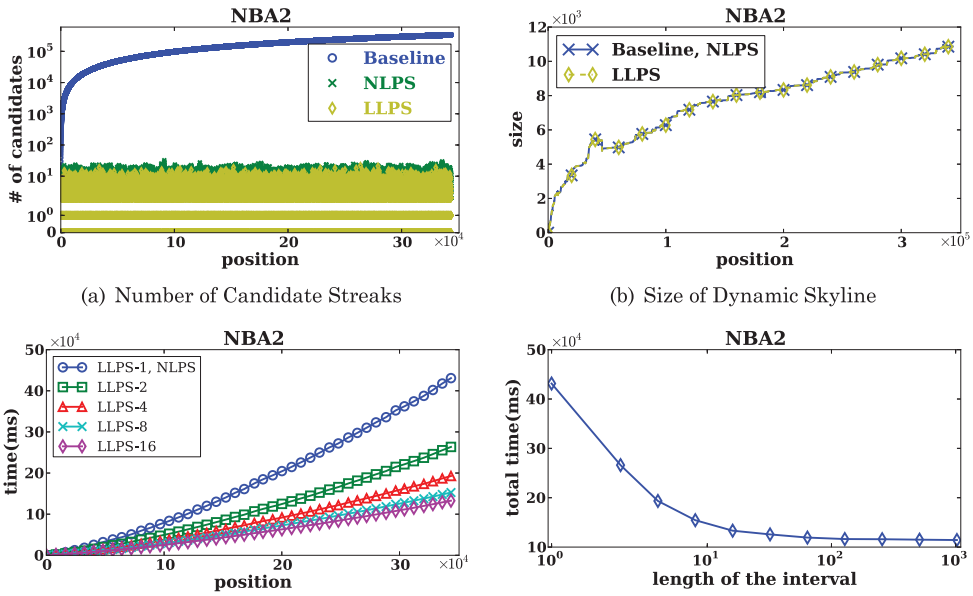


Fig. 14. Distribution of prominent streaks by length.



(a) Number of Candidate Streaks

(b) Size of Dynamic Skyline

(c) Cumulative Execution Time at Various Positions, for Different Reporting Frequencies

(d) Total Execution Time by Reporting Frequencies

Fig. 15. Cumulative execution time at various positions for different reporting frequencies, multidimensional prominent streak discovery.

Prominent streak discovery provides insightful data patterns for data analysis in many real-world applications and is an enabling technique for computational journalism. Given its real-world usefulness and complexity, the research on prominent streaks in sequence data opens a spectrum of challenging problems. Here we briefly outline several future directions. First, a more general concept of prominent streak can be pursued. For instance, finding conditional prominent streaks is about discovering constrains that make streaks prominent (e.g., "since June 2009" and "the month of July" for the motivating example streaks in Section 1.) Second, prominent streaks can be incorporated with the model of data cube [Gray et al. 1997]. Specifically, given a multidimensional sequence, the goal is to discover prominent streaks in not only the full

space but also all possible subspaces. For example, given the NBA2 dataset used in our experiments, we may want to find prominent streaks in spaces (points, rebounds), (points, assists, blocks), and so on. Third, when there are many prominent streaks, it is important to rank them by their interestingness so that a user can focus on the top-ranked prominent streaks. Some important ranking criteria to consider include streak length, number of similar prominent streaks in the dataset, and so on.

ACKNOWLEDGMENTS

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

REFERENCES

Rakesh Agrawal, Christos Faloutsos, and Arun Swami. 1993. Efficient similarity search in sequence databases. *Lecture Notes in Computer Science*, Vol. 730, 69–84. DOI:http://dx.doi.org/10.1007/3-540-57301-1_5

Rakesh Agrawal, King-Ip Lin, Harpreet S. Sawhney, and Kyuseok Shim. 1995. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceedings of the 21th International Conference on Very Large Data Bases (VLDB'95)*. Morgan Kaufmann, San Francisco, CA, 490–501. http://dl.acm.org/citation.cfm?id=645921.673155

Rakesh Agrawal and Ramakrishnan Srikant. 1995. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering*. 3–14.

Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. 1990. Basic local alignment search tool. *Journal of Molecular Biology* 215, 3, 403–410.

Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM* 18, 9 (September 1975), 509–517. DOI:http://dx.doi.org/10.1145/361002.361007

Jon Louis Bentley. 1979. Multidimensional binary search trees in database applications. *IEEE Transactions on Software Engineering* 4, 333–340. DOI:http://dx.doi.org/10.1109/TSE.1979.234200

Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. 2001. The Skyline operator. In *Proceedings of the 17th International Conference on Data Engineering*. 421–430. DOI:http://dx.doi.org/10.1109/ICDE.2001.914855

Chee-Yong Chan, H. V. Jagadish, Kian-Lee Tan, Anthony K. H. Tung, and Zhenjie Zhang. 2006. On high dimensional skylines. In *Proceedings of the 10th International Conference on Advances in Database Technology*. Springer-Verlag, Berlin, 478–495. DOI:http://dx.doi.org/10.1007/11687238_30

Jan Chomicki, Paolo Godfrey, Jarek Gryz, and Dongming Liang. 2003. Skyline with presorting. In *Proceedings of the 19th International Conference on Data Engineering*. 717–719. DOI:http://dx.doi.org/10.1109/ICDE.2003.1260846

Sarah Cohen, Chengkai Li, Jun Yang, and Cong Yu. 2011. Computational journalism: A call to arms to database researchers. In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR'11)*. 148–151.

Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. 1993. *Fast Subsequence Matching in Time-Series Databases*. University of Maryland at College Park, College Park, MD.

Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. 1997. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery* 1, 1 (January 1997), 29–53. DOI:http://dx.doi.org/10.1023/A:1009726021843

Bin Jiang and Jian Pei. 2009. Online interval skyline queries on time series. In *Proceedings of the 2009 IEEE International Conference on Data Engineering (ICDE'09)*. IEEE Computer Society, Washington, DC, 1036–1047. DOI:http://dx.doi.org/10.1109/ICDE.2009.70

Xiao Jiang, Chengkai Li, Ping Luo, Min Wang, and Yong Yu. 2011. Prominent streak discovery in sequence data. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11)*. ACM, New York, NY, 1280–1288. DOI:http://dx.doi.org/10.1145/2020408.2020601

Donald Kossmann, Frank Ramsak, and Steffen Rost. 2002. Shooting stars in the sky: An online algorithm for skyline queries. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02)*. 275–286. http://dl.acm.org/citation.cfm?id=1287369.1287394

H. T. Kung, F. Luccio, and F. P. Preparata. 1975. On finding the maxima of a set of vectors. *Journal of the ACM* 22, 4 (October 1975), 469–476. DOI:http://dx.doi.org/10.1145/321906.321910

T. Warren Liao. 2005. Clustering of time series data—a survey. *Pattern Recognition* 38, 11 (November 2005), 1857–1874. DOI:http://dx.doi.org/10.1016/j.patcog.2005.01.025

Xuemin Lin, Yidong Yuan, Qing Zhang, and Ying Zhang. 2007. Selecting stars: The k most representative skyline operator. In *Proceedings of the IEEE 23rd International Conference on Data Engineering (ICDE'07)*. 86–95. DOI:http://dx.doi.org/10.1109/ICDE.2007.367854

Tim Oates, Laura Firoiu, and Paul R. Cohen. 1999. 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*. 17–21.

Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive skyline computation in database systems. *ACM Transactions on Database Systems* 30, 1 (March 2005), 41–82. DOI:http://dx.doi.org/10.1145/1061318.1061320

Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. 2004. Mining sequential patterns by pattern-growth: The PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering* 16, 11 (November 2004), 1424–1440. DOI:http://dx.doi.org/10.1109/TKDE.2004.77

Jian Pei, Yidong Yuan, Xuemin Lin, Wen Jin, Martin Ester, Qing Liu, Wei Wang, Yufei Tao, Jeffrey Xu Yu, and Qing Zhang. 2006. Towards multidimensional subspace skyline analysis. *ACM Transactions on Database Systems* 31, 4 (December 2006), 1335–1381. DOI:http://dx.doi.org/10.1145/1189769.1189774

Lawrence Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77, 2, 257–286. DOI:http://dx.doi.org/10.1109/5.18626

Young-In Shin and Donald Fussell. 2007. Parametric kernels for sequence data analysis. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI'07)*. Morgan Kaufmann, San Francisco, CA, 1047–1052. http://dl.acm.org/citation.cfm?id=1625275.1625445

Padhraic Smyth. 1997. Clustering sequences with hidden Markov models. In *Advances in Neural Information Processing Systems*. MIT Press, Cambridge, MA, 648–654.

Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology*. 3–17. DOI:http://dx.doi.org/10.1007/BFb0014140

Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi. 2001. Efficient progressive skyline computation. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*. Morgan Kaufmann, San Francisco, CA, 301–310. http://dl.acm.org/citation.cfm?id=645927.672217

Yufei Tao, Ling Ding, Xuemin Lin, and Jian Pei. 2009. Distance-based representative skyline. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*. IEEE Computer Society, Washington, DC, 892–903. DOI:http://dx.doi.org/10.1109/ICDE.2009.84

Yufei Tao, Xiaokui Xiao, and Jian Pei. 2006. SUBSKY: Efficient computation of skylines in subspaces. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*. IEEE Computer Society, Washington, DC, 65–. DOI:http://dx.doi.org/10.1109/ICDE.2006.149

Min Wang and X. Sean Wang. 2006. Finding the plateau in an aggregated time series. In *Proceedings of the 7th International Conference on Advances in Web-Age Information Management (WAIM'06)*. Springer-Verlag, Berlin, 325–336. DOI:http://dx.doi.org/10.1007/11775300_28

Weng-Keen Wong. 2004. *Data Mining for Early Disease Outbreak Detection*. Ph.D. Dissertation. Pittsburgh, PA.

Tian Xia and Donghui Zhang. 2006. 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 (SIGMOD'06)*. ACM, New York, NY, 491–502. DOI:http://dx.doi.org/10.1145/1142473.1142529

Xifeng Yan, Jiawei Han, and Ramin Afshar. 2003. CloSpan: Mining closed sequential patterns in large datasets. In *Proceedings of SIAM International Conference on Data Mining*. 166–177.

Byoung-Kee Yi, H. V. Jagadish, and Christos Faloutsos. 1998. Efficient retrieval of similar time sequences under time warping. In *Proceedings of the 14th International Conference on Data Engineering*. 201–208. DOI:http://dx.doi.org/10.1109/ICDE.1998.655778

Mohammed J. Zaki. 2001. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning* 42, 1–2, 31–60. DOI:http://dx.doi.org/10.1023/A:1007652502315

Zhenjie Zhang, Xinyu Guo, Hua Lu, Anthony K. H. Tung, and Nan Wang. 2005. Discovering strong skyline points in high dimensional spaces. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management (CIKM'05)*. ACM, New York, NY, 247–248. DOI:http://dx.doi.org/10.1145/1099554.1099610

Received February 2013; accepted August 2013