On the brink: Searching for drops in sensor data[*]

Gong Chen
UCLA Statistics
gchen@stat.ucla.edu

Junghoo Cho
UCLA Computer Science
cho@cs.ucla.edu

Mark H. Hansen
UCLA Statistics
cocteau@stat.ucla.edu

ABSTRACT

Sensor networks have been widely used to collect data about the environment. When analyzing data from these systems, people tend to ask exploratory questions—they want to find subsets of data, namely signal, reflecting some characteristics of the environment. In this paper, we study the problem of searching for drops in sensor data. Specifically, the search is to find periods in history when a certain amount of drop over a threshold occurs in data within a time span. We propose a framework, *SegDiff*, for extracting features, compressing them, and transforming the search into standard database queries. Approximate results are returned from the framework with the guarantee that no true events are missed and false positives are within a user specified tolerance. The framework efficiently utilizes space and provides fast response to users' search. Experimental results with real world data demonstrate the efficiency of our framework with respect to feature size and search time.

1. INTRODUCTION

Networks of wireless sensors can record detailed observations about their surroundings. In the context of environmental monitoring, these systems produce rich spatio-temporal data sets. At James Reserve in the San Jacinto mountains, a network of twenty-five wireless sensors, arranged in two parallel lines across a canyon, records air temperature every five minutes. The network is designed to collect data to help studying of the occurrences of so-called Cold Air Drainage (CAD) events. A CAD event involves a sharp drop in temperature in early mornings. The cold air movements can affect plants and animals that may be frost-sensitive or humidity-sensitive, affect the spread of

disease, and set micro-geographic limits on plant distribution. Therefore, it is very important for biologists to study this type of transient atmospheric events.

Biologists would like to search for CAD events in a large collection of recorded data at James Reserve, in particular searching for periods that experience a certain amount of drop in temperature within a time span. When we started our collaboration, we were told that a CAD event involves a drop of no less than 3 degree Celsius within 1 hour. As our interactions developed, it became clear that the biologists needed an exploratory tool, allowing them to pose queries with different drops and time spans. Note that these queries are one time ad-hoc queries to historical data instead of continuous queries; users issue these queries and expect fast response for interaction and exploration purpose rather than for monitoring.

The problem identified above can be formulated as follows: Users want to search for periods in history when data (1 dimensional time series) reflect the event of no less than V units change over T time units. The change V and time span T are both specified by users. Figure 1 (a) shows a day of data from the CAD transect.
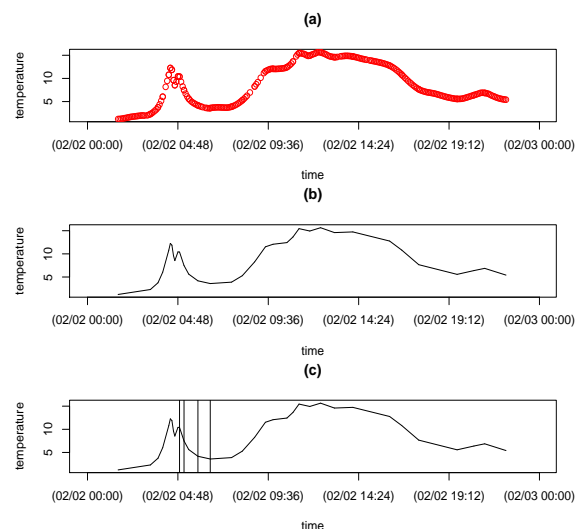

Figure 1: (a) Data; (b) segments: piecewise linear approximation of data; (c) a search result overlaid with segments

A naive approach for solving this problem would be taking the difference between any two observation values within T

---

time units and comparing the differences with V on the fly. Unfortunately, this approach would take several hours for a reasonably large data set to complete a single search. It is too expensive. It is not hard to notice that we can store the differences Δv between any two values between which the time span Δt is within w time units so that a standard range query on Δv and Δt can return the search results much faster than the naive approach as long as $T \leq w$. We call this method the exhaustive search **Exh**. This approach has very large space requirements. It is problematic when the data volume accumulated in history is large. Furthermore, the response time will be degraded by the increasingly large space consumption.

In this paper, we propose the following framework to tackle the search problem: Data are first segmented into a piecewise linear approximation, and then features based on this representation are extracted and stored in a relational database, and finally standard database queries on features are issued to return search results. Figure 1 (b) shows the piecewise linear approximation of the data in (a). Figure 1 (c) shows a search result from our framework. The search result is a tuple of four time stamps indicated by four vertical lines: (from left to right) The first two indicate a drop starting in the period between these two time stamps, and the last two mark the period in which the drop ends. The first two time stamps are two ends of a segment and so are the last two. Once the periods indicated by ends of a pair of segments are found, biologists can further explore the characteristics of data collected in these periods.

The proposed framework, which we refer to as **SegDiff**, efficiently utilizes space and provides fast response to users' search, allowing for exploratory data analysis. Search results are returned with the guarantee that all true events are included and false positives are within a user specified tolerance. Although the application context of our problem is temperature data collected by sensors, the problem statement in Section 2 is generalized to time series data and the framework can work in other contexts.

The paper is organized as follows. In Section 2, we provide the problem statement. In Section 3, we describe the intuitive idea of the proposed framework. In Section 4, we formulate each component of the framework. In Section 5, we present the analysis about quality of results and compression rate. In Section 6, experimental results about performance are presented. In Section 7, we review the related work. We conclude our work in Section 8.

2. DATA GENERATING MODEL AND PROBLEM STATEMENT

In real-world applications, time series always come in at a certain sampling rate. The higher the sampling rate, the more values generated by nature are collected. Since an event (a drop) can happen at the time when no data is being sampled, we need to define a data generating model so that an event is well-defined and reflected by the sampled data.

Definition 1. Data Generating Model G: Let (t_i, v_i) and (t_{i+1}, v_{i+1}) be two consecutive sampled observations in time series, the observation generated by nature is (t, v) where v is defined as follows:

- $v = v_i$ if $t = t_i$;
- $v = v_i + \frac{v_i - v_{i+1}}{t_i - t_{i+1}}(t - t_i)$ if $t_i < t < t_{i+1}$;
- $v = v_{i+1}$ if $t = t_{i+1}$.

In this definition, we assume that the data missing between two sampled observations is produced by the linear interpolation of those two observations. We note that this model is assumed for simplicity of our discussion. In fact, our proposed framework will find all correct events under more general models, as long as the true value from the general model lies within a user-specified error range from the piecewise linear approximation. The exact meaning of this error range will be clarified when we present Lemma 1 in Section 4.1.

Problem Statement Given any two observations (t', v') and (t'', v'') generated by Data Generating Model G, an *event* identified by the time stamps (t', t'') is defined as $(\Delta t, \Delta v)$ where $\Delta t = t'' - t'$ and $\Delta v = v'' - v'$. Given user-specified thresholds T and V, the problem is to search for all events that satisfy the following two constraints for *drop search*: $0 < \Delta t \leq T$ and $\Delta v \leq V < 0$. The two constraints for *jump search* are: $0 < \Delta t \leq T$ and $\Delta v \geq V > 0$. An event satisfying these constraints is a *true event*. We note that any of the two observations (t', v') and (t'', v'') for an event can be either a sampled observation or an observation that is not sampled but produced by Data Generating Model G.

3. OVERVIEW OF THE FRAMEWORK

In this section, we describe the intuitive idea of our framework.

Feature space Let us start by reviewing the search problem. An event that users are searching for consists of two factors: One is a change Δv and the other is a time span of the change Δt. We can map an event into a two-dimensional space shown in Figure 2 (left) with one dimension measuring Δv and the other measuring Δt. Any point in this space is associated with two factors $(\Delta t, \Delta v)$, denoting a potential event. All events reflected in data can be found in this space. We call this space *feature space*.

A query region Next let us review the search conditions. A user's search involves two conditions: One is the threshold for change V so that $|\Delta v| \geq V$ and the other is the threshold for time span of the change T so that $\Delta t \leq T$. In feature space, a user's search can be easily mapped into a region satisfying above two conditions as shown in Figure 2 (middle) when the search is about drops. We call this kind of region *a query region*. The search problem can be reduced to find periods involving at least one event with its mapped point in feature space falling into a query region.
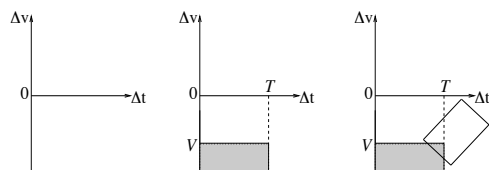

Figure 2: Feature space (left); a query region (middle); intersection between a query region and a parallelogram(right)

Piecewise linear approximation of data With the reduced problem in mind, we need to find an efficient representation of points in feature space so that the detection of points falling into a query region can be quickly answered without considering all potential points reflected by data.

The essence of change Δv and time span Δt naturally leads to linear approximation of data: Each small continuous part of data can be locally approximated by a line *segment*. Data can thus be approximated by many segments (Figure 1 (b)). A segment characterizes the regularity of change Δv and time span Δt in its corresponding part of data.

Parallelogram representation of features Although a segment is a good summary of points in feature space for the part it represents, it is impossible to summarize a point that corresponds to an event occurring across two non-overlapping parts of the data. Figure 3 illustrates this problem. It shows two disjoint parts of the data AB and CD and their corresponding linear approximation—two segments. Point B' is a point in AB and point C' is a point in CD. Suppose that both the drop from C' to B' and its corresponding time span satisfy the search conditions. Then the event featured by such drop and time span is denoted by a point in feature space falling into the query region. A single segment either AB or CD is incapable of characterizing events of this kind (with features reflected by two points on two different segments) simply because information from the other segment is missing. This leads to the motivation of our key representation of features—parallelograms. Figure 4 shows the two segments the same as the ones in Figure 3 and a parallelogram in feature space. This parallelogram is constructed from the two segments by linearly connecting four points in feature space: The point associated with B and C, the point associated with B and D, the point associated with A and D, and the point associated with A and C. The idea is to use events among four ends from two segments to capture all events occurring across these two segments. For example, the event happening at B and C and the event happening at B and D can be used to capture all events occurring at B and any point in CD—the slope of CD is a fixed number. Similar situations exist for other combinations of points. Such a parallelogram is capable of summarizing any point whose associated event occurs across two non-overlapping parts of data (Lemma 3 provides details about this later). In addition, when two segments are from the same part of data, their parallelogram degenerates to a segment, representing any point associated with an event occurring within the part of data.
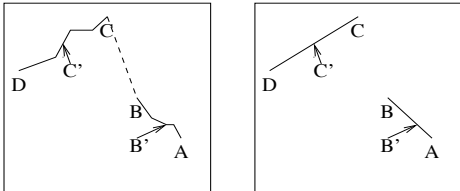

Figure 3: Two disjoint parts (left) and the corresponding segments (right)

Intersection between a parallelogram and a query region With parallelograms summarizing all events in data, we just need to detect intersection between a query region and all parallelograms in feature space. Figure 2 (right) shows an intersection. A returned parallelogram contains at least one point falling into the query region. Therefore, the periods associated with the two segments experience at least one event satisfying the search conditions.

Corner point reduction and range queries for in-

tersection detection Although four corner points uniquely identify the position of a parallelogram, not all four corner points are necessary for intersection detection. As shown in Figure 2 (right), it is sufficient to record two corner points associated with the lower left boundary of the parallelogram to detect the intersection between the query region and the parallelogram for this case. Then simple standard range queries on the coordinates of two corner points can be used to detect intersection. In this way, our framework successfully solves the search problem.

Quality guarantee Since parallelograms are generated from piecewise linear approximation of data, results simply from the above presentation can miss some true events and can include false positives. However, true events can all be discovered by moving parallelograms in feature space, and moreover false positives involved are guaranteed to be within a user specified tolerance . The details are provided by Lemma 4 in Section 4.3.1 and Lemma 5 in Section 5.1.

4. THE FRAMEWORK

In this section, we first formally define the goal of our proposed framework and then present each component of the framework. Our framework provides an approximate solution for the search problem defined in Section 2. Since the approximation involves piecewise linear segmentation of the input time series data, we need to define a metric to measure the quality of the approximation.

Definition 2. Piecewise Linear Approximation: Given data (t, v) generated by Data Generating model G and a user specified error tolerance ϵ where $\epsilon \geq 0$, the piecewise linear approximation of the data is a piecewise linear function f satisfying $|f(t) - v| < \epsilon/2$.

This definition says that a value on a segment from piecewise linear approximation of a time series is bounded within a certain range of the original value.

The goal of our framework is to return any period associated with a pair of segments that experiences at least one drop (or jump) within a certain error tolerance. Specifically, all true events should be found, and all returned false positives should be within a certain error tolerance.

Definition 3. Goal: Given (1) data (t', v') and (t'', v'') generated by Data Generating Model G, (2) a user-specified error tolerance ϵ where $\epsilon \geq 0$, (3) its piecewise linear approximation f, and (4) user-specified thresholds T and V, let t_C and t_D be time stamps of two ends of a segment CD from f, and let t_A and t_B be time stamps of two ends of another segment AB from f, the goal of our framework is to identify all tuples $((t_D, t_C), (t_B, t_A))$ satisfying the following condition: There exists a pair of time stamps (t', t'') so that (1) $t_D \leq t' \leq t_C$ and $t_B \leq t'' \leq t_A$; (2) $0 < \Delta t \leq T$ and $\Delta v \leq V + 2\epsilon$ ($0 < \Delta t \leq T$ and $\Delta v \geq V - 2\epsilon$) for **drop search** (for **jump search**) where $\Delta t = t'' - t'$ and $\Delta v = v'' - v'$.

It is clear that all true events for drop search with $\Delta v \leq V$ will satisfy $\Delta v \leq V + 2\epsilon$ and thus should be found if the goal is achieved by our framework. Any returned false positive is within 2ϵ error tolerance. We note that in our framework an event is not returned by its time stamps (t', t'') but returned in the form of the time stamps of the ends of two segments

involving that event. A returned tuple $((t_D, t_C), (t_B, t_A))$ can involve multiple true events.

Users need to provide an additional fact: What is the longest time span or time window they would be interested in? This fact is pre-defined by users through a constant w. Any $T \leq w$ is supported by our framework. For example, w can be 24 hours if users do not care about any event whose time span is longer than one day. Figure 6 shows a time window in our framework.

With the notations summarized in Table 1, we describe our framework in the following sub-sections.

Notation	Description
T	Threshold for time span
V	Threshold for drop (jump)
t_i	Time stamp of an observation i
v_i	Value at time stamp t_i
Δv_{ij}	$\Delta v_{ij} = v_i$ - v_j where $t_i \geq t_j$
Δt_{ij}	$\Delta t_{ij} = t_i$ - t_j where $t_i \geq t_j$
ϵ	Error tolerance
w	w time units, the width of a time window
r	Compression rate of segmentation
n	Total number of observations

Table 1: Summary of notations used in the paper

4.1 Segmentation and piecewise linear approximation

Since our feature representation is based on piecewise linear approximation of data, we need a segmentation algorithm to achieve the approximation defined in Definition 2. There are many existing algorithms for segmentation. A good review is provided by [5]. We choose one of them for our purpose: The generic online sliding window algorithm is described in Section 2.1 of [5] and linear interpolation is used for approximation. The maximum error for segmentation is $\epsilon/2$ where ϵ is specified by users. That is, the absolute difference between any value of a segment and its corresponding original value of the series should be no greater than $\epsilon/2$. Given a time series (t_0, v_0), (t_1, v_1), ... where $t_i < t_j$ if $i < j$ and a user-defined error tolerance ϵ, the output of segmentation is piecewise linear approximation of the input series. Readers interested in the details of segmentation can refer to [5].

We next show that the segmentation process achieves the approximation in Definition 2.

LEMMA 1. *Let f be the function of piecewise linear approximation output by the segmentation process, (t, v) be the data generated by Data Generating Model G. Then we have*

$$|f(t) - v| \leq \epsilon/2$$

PROOF. The proof is trivial for the case when (t, v) is a sampled observation (t_i, v_i) in the input time series which is ensured by the segmentation process. We next show that the lemma holds when (t, v) is not a sampled observation. Suppose (t, v) is in between two consecutive sampled observations (t_i, v_i) and (t_{i+1}, v_{i+1}) whose corresponding points on a segment are $(t_i, f(t_i))$ and $(t_{i+1}, f(t_{i+1}))$. Then we have $t_i < t < t_{i+1}$. We conduct the proof by contradiction. Assume that $|f(t) - v| > \epsilon/2$. Let us consider the case of $v - f(t) > \epsilon/2$.

$v - f(t) > \epsilon/2. \Rightarrow$
$v_i + \frac{v_{i+1}-v_i}{t_{i+1}-t_i}(t - t_i) - (f(t_i) + \frac{f(t_{i+1})-f(t_i)}{t_{i+1}-t_i}(t - t_i)) > \epsilon/2$
because (1) point (t, v) is on the line interpolated by points (t_i, v_i) and (t_{i+1}, v_{i+1}) and (2) point $(t, f(t))$ is on the segment which contains points $(t_i, f(t_i))$ and $(t_{i+1}, f(t_{i+1}))$. $\Rightarrow$
Equivalently,
$v_i - f(t_i) + \frac{(v_{i+1}-f(t_{i+1}))-(v_i-f(t_i))}{t_{i+1}-t_i}(t - t_i) > \epsilon/2. \Rightarrow$
Case 1: If $(v_{i+1}$ - $f(t_{i+1})) > (v_i - f(t_i))$,
$v_i - f(t_i) + (v_{i+1} - f(t_{i+1})) - (v_i - f(t_i)) >$
$v_i - f(t_i) + \frac{(v_{i+1}-f(t_{i+1}))-(v_i-f(t_i))}{t_{i+1}-t_i}(t - t_i) > \epsilon/2$
since $\frac{t-t_i}{t_{i+1}-t_i} < 1$.
$\Rightarrow v_{i+1} - f(t_{i+1}) > \epsilon/2$. Contradiction achieved.
Case 2: If $(v_{i+1}$ - $f(t_{i+1})) \leq (v_i - f(t_i))$,
$v_i - f(t_i) \geq$
$v_i - f(t_i) + \frac{(v_{i+1}-f(t_{i+1}))-(v_i-f(t_i))}{t_{i+1}-t_i}(t - t_i) > \epsilon/2$
since $\frac{t-t_i}{t_{i+1}-t_i} > 0$.
$\Rightarrow v_i - f(t_i) > \epsilon/2$. Contradiction achieved.
With similar derivation, we can achieve contradiction for $v - f(t) < -\epsilon/2$. Therefore, the lemma is true. $\square$

We note that our framework proposed in the next few sections depends on Lemma 1, which states that the true value from the Data Generating Model G can be different from the piecewise linear approximation at most by $\epsilon/2$. The results from our framework, therefore, is valid even when the Data Generating Model is different from what we assumed here, as long as the true value from the Model differs from the piecewise linear approximation at most by $\epsilon/2$.

Compared to the difference between any two observation values, the difference between two corresponding values on segments is at most off by the user-defined error tolerance ϵ. Lemma 2 formulates this claim.

LEMMA 2. *Let f be the function of piecewise linear approximation output by the segmentation process, (t', v') and (t'', v'') be the data generated by Data Generating Model G.*

$$v' - v'' - \epsilon \leq f(t') - f(t'') \leq v' - v'' + \epsilon$$

PROOF. By Lemma 1, $v' - \epsilon/2 \leq f(t') \leq v' + \epsilon/2$ and $-v'' - \epsilon/2 \leq -f(t'') \leq -v'' + \epsilon/2$. The addition of the two inequalities produces the result of Lemma 2. $\square$

4.2 Feature Representation

In order to present our feature representation scheme, we first define the following concepts. *Feature space* is a space with two orthogonal dimensions: One dimension measures difference $\Delta v_{ij} = (v_i - v_j)$ and the other measures time span $\Delta t_{ij} = (t_i - t_j)$. An event $(\Delta t_{ij}, \Delta v_{ij})$ for two points (t_i, v_i) and (t_j, v_j) is associated with a point in feature space. Such a point is called a *feature point* and is denoted by IJ. A segment that linearly connects two feature points IJ $(\Delta t_{ij}, \Delta v_{ij})$ and $I'J'$ $(\Delta t_{i'j'}, \Delta v_{i'j'})$ in feature space is called a *feature segment* and is denoted by $(IJ, I'J')$. A *query region* is a region in feature space defined by the search thresholds T and V so that $\Delta v_{ij} \leq V < 0$ ($\Delta v_{ij} \geq V > 0$) and $0 < \Delta t_{ij} \leq T$ for drop search (for jump search). To differentiate a feature segment and a segment output by the segmentation process, we call the latter as a *data segment*. A data segment ES is defined by $((t_s, v_s), (t_e, v_e))$ where (t_s, v_s) is the start observation represented by that segment and (t_e, v_e) the end observation.

We design *feature parallelograms* to represent feature points used for search. A feature parallelogram essentially summarizes feature points that could be produced by taking differences between any two points on two data segments.
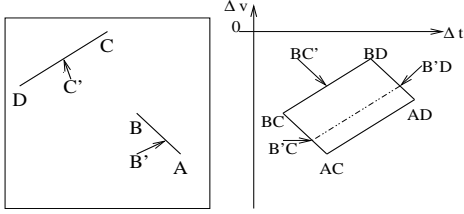

Figure 4: Two data segments (left) and the corresponding parallelogram in feature space (right)

LEMMA 3. *Given two data segments CD ((t_D, v_D), $(t_C, -v_C)$) and AB ((t_B, v_B), (t_A, v_A)) output by the segmentation process where $t_B \geq t_C$, (1) the quadrangle formed by linearly connecting the four feature points BC (Δt_{BC}, Δv_{BC}), BD (Δt_{BD}, Δv_{BD}), AD (Δt_{AD}, Δv_{AD}), and AC (Δt_{AC}, Δv_{AC}) in feature space is a parallelogram and (2) the region of this parallelogram (BC, BD, AD, AC) captures all feature points associated with any two points with one point on data segment AB and the other on data segment CD.*

PROOF. We first prove that the quadrangle (BC, BD, AD, AC) is a parallelogram. Then we consider points on boundaries of a parallelogram and describe the properties of feature segments inside a parallelogram. Finally we use these results to prove the second part of Lemma 3.

Figure 4 shows two data segments and their corresponding feature parallelogram in feature space. In this figure, the order $v_A < v_B < v_D < v_C$ is irrelevant to the following derivations but is included here for descriptive purpose.

A parallelogram Feature segment (BC, BD) has the slope $\frac{\Delta v_{BD} - \Delta v_{BC}}{\Delta t_{BD} - \Delta t_{BC}}$ where the numerator is $(v_B - v_D) - (v_B - v_C) = v_C - v_D$ and the denominator is $(t_B - t_D) - (t_B - t_C) = t_C - t_D$. This shows that feature segment (BC, BD) has the same time span and the same slope as data segment CD in the left. By following the same derivation, we can show that feature segment (AC, AD) has the same time span and the same slope as data segment CD. Similarly, we can show that feature segments (BC, AC) and (BD, AD) have the same time span and the same slope as data segment AB in the left. Therefore, feature segments (BC, BD), (AC, AD), (BC, AC) and (BD, AD) form a parallelogram in feature space. We use (BC, BD, AD, AC) to denote the parallelogram.

Points on boundaries of a parallelogram We examine a point C' on data segment CD in the left of Figure 4 and its related feature points in the right.

Feature segment (BC, BC') has the same slope as feature segment (BC, BD) because (1) $\frac{\Delta v_{BC'} - \Delta v_{BC}}{\Delta t_{BC'} - \Delta t_{BC}} = \frac{(v_B - v_{C'}) - (v_B - v_C)}{(t_B - t_{C'}) - (t_B - t_C)} = \frac{v_C - v_{C'}}{t_C - t_{C'}} = \frac{v_C - v_D}{t_C - t_D}$ where the last quantity is the slope of data segment CD and (2) data segment CD has the same slope as feature segment (BC, BD). The last equality in (1) holds since segment CC' has the same slope as data segment CD.

Since (1) feature segment (BC, BC') has the same slope as feature segment (BC, BD) and (2) $t_D \leq t_{C'} \leq t_C$ (C' is on data segment CD) and $\Delta t_{BC} \leq \Delta t_{BC'} \leq \Delta t_{BD}$, feature point BC' representing ($\Delta t_{BC'}$, $\Delta v_{BC'}$) is on feature

segment (BC, BD). As C' is an *arbitrary* point on data segment CD, feature segment (BC, BD) summarizes the feature point ($\Delta t_{BC'}$, $\Delta v_{BC'}$) associated with point B and an arbitrary point C' on data segment CD.

Similarly, feature point $B'C$ is on feature segment (BC, AC) where point B' is on data segment AB. Feature segment (BC, AC) summarizes the feature point associated with point C and an arbitrary point B' on data segment AB.

Feature segments inside the parallelogram We claim that feature segment ($B'C$, $B'D$) must summarize feature points associated with point B' and an arbitrary point on data segment CD. Let us consider the situation where data segment AB is scaled (or shrunk) to AB' with the same slope. In this case, feature segment (BC, BD) is moved to feature segment ($B'C$, $B'D$) and the parallelogram (BC, BD, AD, AC) is scaled (or shrunk) to the parallelogram ($B'C$, $B'D$, AD, AC). As we have shown above that feature segment (BC, BD) summarizes the feature point associated with point B and an arbitrary point C' on data segment CD, feature segment ($B'C$, $B'D$) summarizes the feature point associated with point B' and an arbitrary point C' on data segment CD. Thus, the claim is true.

Let us next think about the above situation inversely: If segment AB' is continuously scaled from segment AA (of zero length) to data segment AB, feature segment ($B'C, B'D$) moves from feature segment (AC, AD) to feature segment (BC, BD) and it summarizes the feature point associated with an *arbitrary* point B' on data segment AB and an arbitrary point C' on data segment CD. The region of parallelogram (BC, BD, AD, AC) is swept by feature segment ($B'C, B'D$) in this movement. Therefore, the second part of Lemma 3 is true. □

We note that a parallelogram degenerates to a feature segment in feature space if the two data segments for its construction are the same, that is, they are from the same part of data. The degenerated parallelogram in feature space summarizes any point whose associated event occurs within the part of data. Therefore, a parallelogram can be used to summarize any event occurs either across two data segments or within a data segment.

4.3 Feature Extraction

4.3.1 Feature Reduction of Parallelograms

If four corner points are collected for a feature parallelogram, they uniquely define the parallelogram and thus they can be used to detect whether a parallelogram intersects a query region. But some corner points are redundant for detecting intersection because of the shapes of the query regions and the parallelograms. Since a parallelogram's shape is determined by the slopes of its two corresponding data segments, we enumerate all possible cases of two slopes to find necessary corners for detecting intersection. Essentially, **a query region of drop (jump) search must intersect the lower (upper) left boundary of a feature parallelogram if it intersects the parallelogram.** Suppose AB and CD are two data segments output by the segmentation process and their slopes are k_{AB} and k_{CD}. All possible cases are listed in Table 2. Let us examine the drop search in *case 1*.

Since data segment AB and data segment CD can denote different absolute values (say, temperature), the parallelo-

Case	Slopes	Type	Corners
1	$k_{CD} \geq 0,$	drop	BC, AC
	$k_{AB} \leq 0$	jump	BC, BD
2	$k_{CD} \geq 0,$	drop	BC
	$k_{AB} \geq k_{CD}$	jump I	BC, AC, AD
		jump II	AD, AC
3	$k_{CD} \geq 0,$	drop	BC
	$0 < k_{AB} < k_{CD}$	jump I	BC, BD, AD
		jump II	AD, BD
4	$k_{CD} < 0,$	drop	BC, BD
	$k_{AB} \geq 0$	jump	BC, AC
5	$k_{CD} < 0,$	drop I	BC, AC, AD
	$k_{AB} \geq k_{CD}$	drop II	AC, AD
		jump	BC
6	$k_{CD} < 0,$	drop I	BC, BD, AD
	$k_{CD} < k_{AB} < 0$	drop II	BD, AD
		jump	BC

Table 2: Necessary corners of a parallelogram constructed from data segments AB and CD for detecting intersection

gram can move around in feature space with varying size. But as long as the two slopes satisfy the condition of this case, the relative positions of four boundaries of a parallelogram hold as demonstrated in Figure 5 by its construction in Lemma 3. Specifically, feature segment (BC, AC) and feature segment (BD, AD) cannot exchange their positions since (1) $\Delta t_{BC} \leq \Delta t_{BD}$ and (2) $k_{CD} \geq 0$ and (3) $\Delta v_{BC} \leq \Delta v_{BD}$. Similarly, feature segment (AC, AD) and feature segment (BC, BD) cannot exchange their positions. Therefore, feature segment (BC, AC) is always the lower left boundary of the parallelogram.
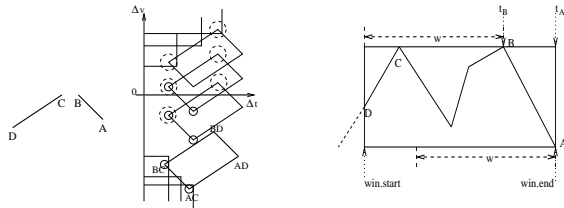

Figure 5: Boundary conditions Figure 6: A time of case 1 window

As shown in Figure 5, if a query region intersects a parallelogram with this kind of shape, one of the following three sub-cases must hold: Feature point BC falls into the region; feature point AC falls into the region; part of feature segment (BC, AC) is inside the region but neither feature point BC nor feature point AC is in the region. Feature segment (BC, AC) is the lower left boundary of the parallelogram. Therefore, we only need to record the features associated with feature points BC and AC to represent that boundary and detect intersection. The solid circles in Figure 5 mark the two points for each parallelogram. Similarly, the broken-line circles in Figure 5 mark the two feature points BC and BD, the corners of the upper left boundary of each parallelogram to support jump search. The corners in the remaining five cases can be founded in the same manner. The details are provided in the Appendix.

The case analysis in Table 2 determines the corner points whose features should be collected according to different

slopes of two data segments for detecting intersection between a query region and a parallelogram. But due to segmentation errors, some true events in data may be missed by the above detection. The difference between two points on data segments is at most ϵ off from the difference between their corresponding original values by Lemma 2. Since the differences captured by a parallelogram are for two points on data segments, they are at most ϵ off from the original values. Lemma 4 provides the solution for capturing all true events.

LEMMA 4. *If all feature parallelograms are shifted down (up) by ϵ in feature space, intersection regions between a query region for drop (jump) search and all parallelograms capture all true events.*

PROOF. Suppose $(\Delta t, \Delta v')$ is a feature point in a parallelogram and its corresponding original feature is $(\Delta t, \Delta v)$. Suppose $\Delta v \leq V < 0$ and $\Delta t \leq T$. Then for drop search, this feature indicates a true event. By the condition $\Delta v \leq V$, we have $\Delta v + \epsilon \leq V + \epsilon$. By Lemma 2, $\Delta v' \leq \Delta v + \epsilon$. Then, we have $\Delta v' \leq V + \epsilon$ and equivalently $\Delta v' - \epsilon \leq V$. By shifting a parallelogram down by ϵ, the feature point $(\Delta t, \Delta v')$ is shifted to $(\Delta t, \Delta v'')$ where $\Delta v'' = \Delta v' - \epsilon$. From the above derivation, $\Delta v'' \leq V$. Therefore, for drop search, a true event is never outside intersection region between a query region and all shifted parallelograms. By symmetry, the same conclusion can be achieved for jump search by shifting all parallelograms up by ϵ. The lemma is proved. □

We are ready to state specific features to be collected for case 1 in Table 2. To guarantee that no event in data is missed due to segmentation errors, SegDiff collects features as follows: If $\Delta v_{AC} - \epsilon \leq 0$, the features $(\Delta t_{BC}, \Delta v_{BC} - \epsilon)$ and $(\Delta t_{AC}, \Delta v_{AC} - \epsilon)$ are collected; if $\Delta v_{BD} + \epsilon > 0$, the features $(\Delta t_{BC}, \Delta v_{BC} + \epsilon)$ and $(\Delta t_{BD}, \Delta v_{BD} + \epsilon)$ are collected. The first condition checks whether a parallelogram contains any drop; the second checks whether a parallelogram contains any jump. By Lemma 2 and Lemma 4, the conditions consider the worst scenario: The difference between two points on the data segments is ϵ off from the difference in an original event, and thus parallelograms are shifted down (up) by ϵ for drop (jump) search to capture this kind of events. We notice that the above scheme will bring false positives. We can, however, guarantee that one returned drop (jump) is always within 2ϵ of drop (jump) in true events. The analysis is presented in Section 5. Specific features to be collected for other cases can be given in the same manner as above.

In the case analysis shown by Table 2, we notice that at most three corner points are needed (for example, in the case of jump search of case 2), and in some case only one corner point is enough (for example, in the case of drop search of case 2). The expected number of corner points depends on the case distribution in data segments.

4.3.2 The Procedure

Algorithm 1 describes how SegDiff extracts features in an online manner. For each newly generated data segment, the features between this data segment and other data segments in a time window are computed. Figure 6 shows a time window. When the start time t_D of data segment CD is earlier than $win.start$, the data segment is truncated at the time $win.start$ (line 4) and is treated to start

---
Algorithm 1 FeatureExtraction

Input Data segments $((t_s, v_s), (t_e, v_e)) \ldots$ output by the segmentation process where the data segments are in temporal order, that is, the end point of a previous input data segment is always the start point of the current input data segment; a user-defined error tolerance ϵ; a user-defined width of a time window w

Output Database tables containing features to support search

1: **while** a data segment $((t_s, v_s), (t_e, v_e))$ newly generated from the segmentation process **do**
2:   $t_B \leftarrow t_s; v_B \leftarrow v_s; t_A \leftarrow t_e; v_{t_A} \leftarrow v_e$
3:   the end time of a time window $win.end \leftarrow t_A$
4:   the start time of a time window $win.start \leftarrow win.end - (t_A - t_B) - w$
5:   **for** each previous data segment CD $((t_D, v_D), (t_C, v_C))$ within the window defined by $(win.start, win.end)$ **do**
6:     according to the description in Section 4.3.1, features are collected for intersection detection with the input parameters of data segment CD and data segment AB as $(t_D, v_D), (t_C, v_C), (t_B, v_B), (t_A, v_A)$
7:   **end for**
8: **end while**
---

from $win.start$ (line 5). The window with the newly defined width ($win.end - win.start$) includes all data segments CD's which represent points whose time stamps are within w of the time stamp of any point on data segment AB. Therefore, all possible events with end time between t_A and t_B and time span no greater than w are captured by parallelograms constructed from data segment AB and every data segment CD within the window.

Both the segmentation process and Algorithm 1 are online processing on their inputs. The features can thus be extracted as soon as data are being collected or uploaded. The benefit is that there is no considerable delay for users to search new data in terms of feature extraction process.

4.4 Queries

We can use two simple range queries to retrieve boundary time points $((t_D, t_C), (t_B, t_A))$ for two data segments CD and AB containing at least one event by checking intersection between a query region and extracted features. The intersection falls into two types: One is for detecting whether a feature point is in a query region; the other is for detecting whether a feature segment (a boundary of a parallelogram) with its two ends outside the region intersects a query region. We call the first type *point query* and the second type *line query*. For example, to perform drop search for case 1 in Figure 5, SegDiff only needs the union of the results of two point queries and one line query: One point query is for checking feature point BC, the other is for checking feature point AC and the line query is for checking feature segment (BC, AC). We list these two types of queries in the context of drop search.

Point query Given a feature point $(\Delta t, \Delta v)$, the following conditions are used to detect whether it is in the query region: $\Delta t \le T$ and $\Delta v \le V$. B-tree index can be built on the concatenation of Δt and Δv.

Line query Given a feature segment $((\Delta t', \Delta v'), (\Delta t'', \Delta v''))$ where $\Delta t' \le \Delta t''$, the following conditions are used to detect whether its two ends are outside the region and whether it intersects a query region: $\Delta t' \le T$ and $\Delta v' > V$ and $\Delta t'' > T$ and $\Delta v'' < V$ and $\Delta v' + \frac{\Delta v'' - \Delta v'}{\Delta t'' - \Delta t'}(T - \Delta t') \le V$. B-tree index can be built on the concatenation of $\Delta t'$, $\Delta v'$, $\Delta t''$, and $\Delta v''$.

5. ANALYSIS

5.1 Approximation

A false positive can be at most 2ϵ different from the original event. One ϵ is from errors in segmentation and the other is from shifting parallelograms.

LEMMA 5. *A pair of data segments returned by SegDiff contains at least one event with drop $\Delta v \le V + 2\epsilon$ (jump $\Delta v \ge V - 2\epsilon$) and time span $0 < \Delta t \le T$ where V and T are user-specified thresholds for drop (jump) search.*

PROOF. By the queries specified by in Section 4.4, the parallelogram of a pair of data segments returned for drop search at least contains a feature point $(\Delta t, \Delta v'')$ so that $0 < \Delta t \le T$ and $\Delta v'' \le V$. Suppose its original feature is $(\Delta t, \Delta v')$ before a parallelogram is shifted. Since the parallelogram is shifted down ϵ to exclude all false negatives, $\Delta v' = \Delta v'' + \epsilon$. Suppose the original event in the data has feature $(\Delta t, \Delta v)$, $\Delta v \le \Delta v' + \epsilon$ by Lemma 2. Combining the equality $\Delta v' = \Delta v'' + \epsilon$ and two inequalities $\Delta v \le \Delta v' + \epsilon$ and $\Delta v'' \le V$, we have $\Delta v \le \Delta v'' + 2\epsilon \le V + 2\epsilon$. □

Theorem 1 states the search quality guarantee of SegDiff.

THEOREM 1. *No true event is missed by SegDiff; false positives have the property of drop $\Delta v \le V + 2\epsilon$ (jump $\Delta v \ge V - 2\epsilon$) and time span $0 < \Delta t \le T$ where V and T are user-specified thresholds for drop (jump) search.*

PROOF. Since SegDiff records the features associated with parallelograms that are shifted down (up) by ϵ for drop (jump) search, the condition of Lemma 4 is satisfied. By that lemma, intersection between a query region and parallelograms captures all true events. The queries specified in Section 4.4 for detecting above intersection are issued to get any pair of data segments whose feature parallelogram intersects a query region. Therefore, the first part of the theorem is true. The second part of the theorem is true by Lemma 5. □

We note that given the data of *Data Generating Model G* in Definition 1, it is impossible for the exhaustive search Exh to find all true events because it only considers the differences between the sampled observations. As we mentioned, a drop can happen when no data is being sampled. Exh cannot capture events of this type. Since all proofs in our framework hold for any data generated by G, the search performed in our framework does not differentiate sampled observations and unobserved ones, and thus can always return all true events reflected by G.

5.2 Compression

SegDiff uses less space than Exh for the following two reasons. (1) In each time window, SegDiff computes features between each newly generated data segment and all data segments in a time window, so the number of features in a time window is proportional to *the number of data segments in that window*. The number of features in a window for Exh, however, is proportional to *the number of data*

points in that window. (2) The number of time windows for SegDiff's feature extraction is proportional to *the total number of data segments* while the number of windows for Exh's is proportional to *the total number of data points*. The first gain is denoted by the term $\frac{n_w}{m_w}$ where n_w is the number of data points in a time window of width w and m_w is the number of data segments in a time window with the width defined in Section 4.3.2. The second gain is denoted by the *compression rate of segmentation r* which is defined as the number of observations represented by one data segment on average. These two gains are achieved by piecewise linear approximation of data and parallelogram representation of features.

In addition, SegDiff does not record all four corner points for each feature parallelogram but only necessary ones. The term $\frac{c1}{c2}$ is used to denote this reduction where c_1 is the number of columns for a database table in Exh and c_2 is the one in SegDiff. $c_1 = 3$ since one row has time span, difference, and an absolute time stamp for uniquely identifying an event. $5 \leq c_2 \leq 7$ and the exact value is case-dependent. In the cases of one corner point, $c_2 = 5$ with two columns as time span and difference, and three columns as absolute time stamps for uniquely identifying two data segments (the fourth absolute time stamp can be computed on the fly). In the cases of two corner points, $c_2 = 6$. In the cases of three corner points, $c_2 = 7$.

The space saving in feature size by using SegDiff in comparison with Exh is $\left(\frac{c_1}{c_2}\frac{n_w}{m_w}r\right)$. That is, Exh uses $\left(\frac{c_1}{c_2}\frac{n_w}{m_w}r\right)$ times space as much as SegDiff does.

We note that m_w is not a constant by its definition, and r is a simple estimate of the number of observations represented by one data segment. Therefore, although the above analysis sheds lights on the space saving of SegDiff in comparison with Exh, it is important to evaluate their empirical performance.

6. EXPERIMENTS

We investigate the performance of our approach SegDiff and the exhaustive search Exh in different settings with the data collected by the Cold Air Drainage Transect from December, 2005 to November, 2006. The data are preprocessed by a smoothing method with robust weights so that anomalies are removed. A subset of data is used in Section 6.1, 6.2, and 6.4 for experimentation efficiency. All data are used in Section 6.3.

All experiments are conducted on a dedicated computer with an Intel Core Duo 2.0 GHz processor, 2 gigabyte 667 DDR2 SDRAM, and a 100GB 1.5Gps SATA disk. The operating system is Mac OS X 10.4.9. MySQL 5.0.37 database implementation is used for feature storage. Standard SQL queries are used for retrieval. Each experiment is repeated 10 times and average values are reported. The default parameter settings are $\epsilon = 0.2$, $w = 8$ hours, $T = 1$ hours and $V = -3$ degree Celsius if not explicitly mentioned. In Section 6.1, 6.2 and 6.3, operating system cache is flushed before every query. The situation where system cache is available is studied in Section 6.4. In that situation, indexes and previously hit disk blocks can remain in memory. Query cache and key cache in database are turned off for all experiments.

We use feature size and disk size to measure space usage. Disk size is the sum of feature size and index size. Query execution time by sequential scan and query execution time using indexes are used to measure time efficiency.

6.1 Performance with different tolerances

Compression rates with different tolerances Table 3 summarizes different compression rates under different error tolerances. The range of drops in this data set is from 0 to -35 degree Celsius. Therefore, $\epsilon = 1$ degree Celsius is a reasonably tight tolerance. For the query 3 degree drop within 1 hour, $\epsilon = 0.2$ may be good enough for users since it provides the guarantee that a pair of returned data segments contains at least one event with at least 2.6 degree drop within 1 hour and no true event is missed. From Table 3, we can see that when the tolerance becomes larger, which leads to less data segments produced, the compression rate becomes higher.

ϵ	0.1	0.2	0.4	0.8	1.0
r	4.73	7.03	10.52	16.10	18.55

Table 3: Compression rate r under different segmentation error tolerances

Feature size with different compression rates We next examine how much space SegDiff uses with different degrees of approximation. Figure 8 shows that the feature size is reduced when the compression rate r increases and the curve has the shape of r^{-1}. This follows our analysis in Section 5.2: The number of data segments is inversely proportional to r and so is the total number of windows. The features generated by Exh is about 383 megabytes, 12 times larger than the size (about 32 megabytes) of features generated by SegDiff for $\epsilon = 0.2$ and $r = 7.03$.
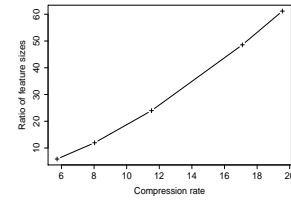

Figure 7: Ratio of feature sizes with different r's

Figure 7 shows that SegDiff gains one order of magnitude of space saving with the compression rate r greater than 7. If a query involves a larger magnitude of drop, a larger ϵ is admissible and orders of magnitude of space saving can be achieved by SegDiff.

Disk size with different compression rates Figure 9 shows a similar trend to the one in Figure 8. Comparing the numbers on these two figures, we can see that the overhead of B-tree indexes for SegDiff is non-trivial, about 1.1 times as large as feature size. This overhead comes from B-tree indexes on multiple columns. Following the way described in Section 4.4, some columns are repeatedly involved in index building process. This makes index size larger than feature size. The size of B-tree indexes for Exh is about half of feature size.
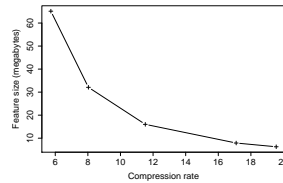

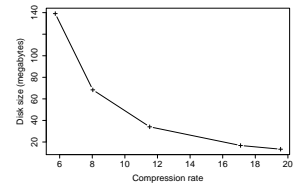**Figure 8: Feature sizes with different r's (Exh's feature size: 383 megabytes)**

Figure 9: Disk sizes with different r's (Exh's disk size: 592 megabytes)

Corner cases Table 4 reports corner case distributions under different error tolerances. For example, when $\epsilon = 0.2$, 19.83% cases need only one corner to support queries. By taking the average $1 \times 19.83\% + 2 \times 46.79\% + 3 \times 33.37\% = 2.13$, we effectively have two corner points cases. It means that the case analysis from Section 4.3.1 effectively reduces the storage of parallelograms' corners by half. This is true for all other ϵ's.

ϵ	0.1	0.2	0.4	0.8	1.0
one corner	17.05	19.83	22.67	25.88	26.90
two corners	46.43	46.79	47.09	47.25	47.10
three corners	36.52	33.37	30.24	26.87	26.00

Table 4: The percentage of different corner cases under different error tolerances

Query execution time with different compression rates Figure 10 shows the sequential scan time decreases when compression rate increases in the same manner as that in Figure 8. Figure 11 shows the similar situation for execution time using indexes. We see that indexes do not help in the case of the query of 3 degrees drop within 1 hour in both approaches: The execution time using indexes is much slower than the sequential scan time. As we will see later in Section 6.4, this specific query falls into a hard region for both approaches in feature space where it retrieves a large number of tuples, making the indexing access inefficient.
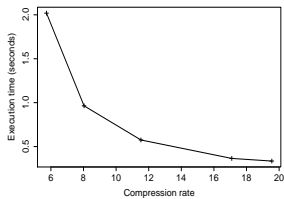
Figure 10: Sequential scan time with different r's (Exh's time: 6.44 seconds)
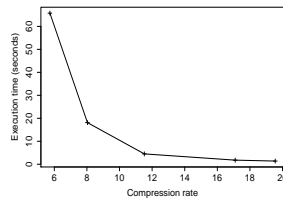
Figure 11: Execution time using indexes with different r's (Exh's time: 386.77 seconds)

Ratio of execution time Table 5 lists space saving r_f and the time gain r_{st} for sequential scan. Table 6 shows the situation when indexing is used. With indexes, the performance difference in time becomes even larger. For $\epsilon = 0.2$, Exh's query execution time is 6.69 times as long as SegDiff's for sequential scan but the former is 21.35 times as long as the latter by using index. Since the number of features from Exh can be an order of magnitude larger than the one from SegDiff, B-tree indexes can be much taller than the ones for SegDiff and thus Exh becomes even slower. This, again, demonstrates the strength of the compression design in SegDiff.

ϵ	0.1	0.2	0.4	0.8	1.0
r_f	5.88	11.95	23.96	48.57	61.71
r_{st}	3.19	6.69	11.20	17.65	19.22

Table 5: Ratio of feature sizes r_f and ratio of sequential scan time r_{st} with ϵ varied

6.2 Performance with different window sizes

ϵ	0.1	0.2	0.4	0.8	1.0
r_d	4.26	8.66	17.37	35.33	44.42
r_{it}	5.88	21.35	85.93	217.00	279.34

Table 6: Ratio of disk sizes r_d and ratio of execution time using indexes r_{it} with ϵ varied
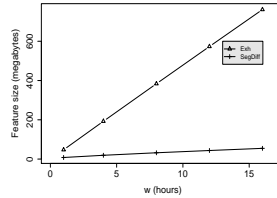

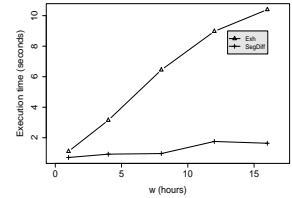**Figure 12: Feature size with w varied**

Figure 13: Sequential scan time with w varied

We fix $\epsilon = 0.2$ to evaluate the impact of window sizes w on performance of SegDiff and Exh. Figure 12 shows that feature sizes of both approaches appear to grow linearly with w. However, Table 7 shows the ratio of feature sizes r_f actually increases with w. There is an order of magnitude difference when w is 8 hours. This is because the number of observations in a window n_w increases almost linearly as w increases, but the number of data segments in a time window m_w does not necessarily grow linearly as w increases—linear growth is the worst case when every two consecutive points are connected by a data segment. The similar situation exists for disk sizes in Table 7. In this case, when users want a system to support queries with larger time spans, SegDiff's advantage becomes more significant. Figure 13 shows the trend of the sequential scan time for the query follows the same pattern as shown in Figure 12.

w	1	4	8	12	16
r_f	5.89	9.98	11.97	13.14	13.94
r_d	4.51	7.30	8.66	9.53	10.18

Table 7: Ratio of feature sizes r_f and ratio of disk sizes r_d with w varied

6.3 Performance with the increasing number of observations

Sensors often continuously collect data for a long period of time and accumulate a large volume of data. It is important to examine the scalability of SegDiff with the increasing number of observations. We split data into 5 groups. Storage and query execution time is checked after one group of data's features are incrementally inserted into the database. It would take too much time to complete Exh's experiments so we abort them after the second groups' features are inserted into the database. But the results still show us how two approaches behave when the number of observations n goes large. Figure 14 shows that the feature size of SegDiff grows almost linearly with n. This is reasonable: The number of windows is the total number of observations n divided by the compression rate r, and r is assumed to be about the same when the error tolerance ϵ is fixed.

Space usage For the first two groups with complete experiments, the space saving r_f is 12.26: With 108 megabytes, SegDiff can handle these two groups while Exh needs 1,328 megabytes. With one quarter of 1,328 megabytes, SegDiff
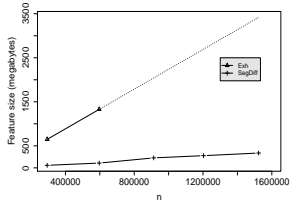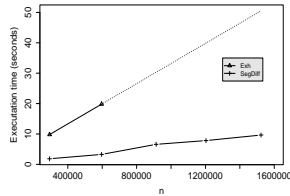
Figure 14: Feature size with n increased

Figure 15: Sequential scan time with n increased

Figure 17: Exh's sequential scan time with cache

Figure 18: SegDiff's sequential scan time with cache

can handle all groups' data while the estimate of Exh's feature size is about 3,416 megabytes. The estimates of Exh's feature size are marked by the dotted line in Figure 14. The analysis in Section 5.2 shows that Exh's feature size should grow linearly with n when window size is fixed, and thus we can get estimates by extrapolating the line of two observed results in Figure 14. The disk size of Exh is an order of magnitude larger than SegDiff's disk size for handling two groups.

Execution time Figure 15 shows that sequential scan time grows almost linearly with n and SegDiff can return results for all sensors within 10 seconds. As for execution time using indexes, Exh is 18 times slower than SegDiff for handling two groups.

6.4 Performance with different query regions

We investigate the query time of SegDiff and Exh with random queries. Figure 16 shows the coverage of these random queries. We first examine the situation where system cache is available. This presents the case where both previously hit disk blocks and indexes can remain in memory.
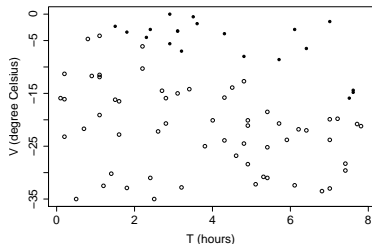

Figure 16: The coverage of random queries

Sequential scan with cache available Figure 17 and Figure 18 share a similar pattern in sequential scan time and the horizontal lines mark boundaries for hard queries in both figures. Among them, the ones that are hard for both approaches are denoted by solid dots in Figure 16. As we can see, the hard area is at the top right triangular region. This is what we expected: The larger a query region is, the more results it retrieved; both approaches have to take longer time.

Indexing with cache available Figure 19 and Figure 20 show execution time using indexes. Again, a similar pattern exists in these two figures but with SegDiff's time shifting to a much lower level.

Since SegDiff compresses events into efficient parallelogram representation and returns data segments, which summarize results, it has much faster response time. Figure 21 and Figure 22 suggest that SegDiff is about 9 times faster than Exh with sequential scan and is about 10 times faster using indexes.
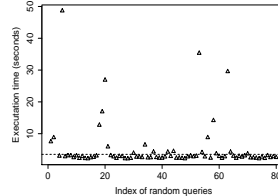
Ratio of execution time without caching Figure 23 and
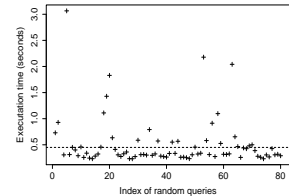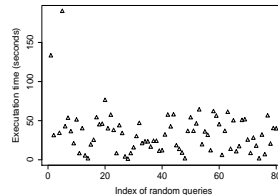

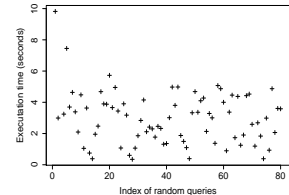**Figure 19: Exh's execution time using indexes with cache**

Figure 20: SegDiff's execution time using indexes with cache

Figure 24 show the performance gain when system cache is not available: SegDiff is about 9 times faster than Exh but is about 20 times faster than Exh using indexes. This illustrates that large indexes hurt the performance when Exh accumulates too many features.

7. RELATED WORK

Compared to many existing work on similarity search in time series database like [1, 12, 4] (readers can refer to [3] for a comprehensive review) that require users to specify a query series, the generic conditions for drop search are more appropriate for users who do not have a precise specification about shape and absolute magnitude of series they are looking for, since numerous series satisfying the conditions can have very different shapes and absolute magnitudes. For similarity search, the studies like [1, 4, 7] map data into boxes and use spatial access method like R*-tree index on these boxes to speed up queries. Our framework uses a different idea of boxing: SegDiff compresses all features, which otherwise have to be enumerated, into parallelograms and stores necessary corner points to support search.

As commented in [5], piecewise linear approximation may be the most frequently used representation of time series data. It is utilized in a variety of settings such as clustering, classification, characterizing movement patterns and shape-search [6, 9, 13, 14]. But to our best knowledge, no existing work uses such representation to construct parallelograms for compressing signal features as we do. We invent a novel usage of piecewise linear approximation.

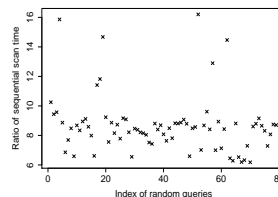There has been a growing interest in burst, novelty, or

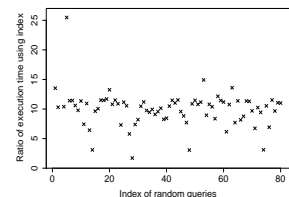**Figure 21: Ratio of sequential scan time with cache**

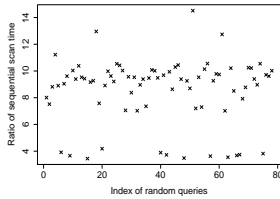**Figure 22: Ratio of execution time using indexes with cache**

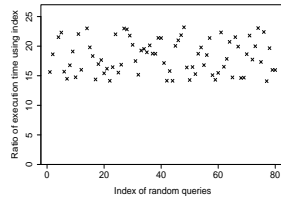**Figure 23: Ratio of sequential scan time without cache**

Figure 24: Ratio of execution time without cache

change detection in time series. Zhu and Shasha [15] consider online burst detection problem as discovering summation of time series in a sliding window with each known size greater than each corresponding known threshold. They focus on improving detection time in an online monitoring setting. Drops targeted by our framework are differences in time series and time span of occurrences is not fixed. Although features are collected by online procedures in order to support timely update in database, our application context is one time search. Ma and Perkins [8] employ support vector regression to predict future values' confidence interval and a change occurs when it falls outside the interval. Reznik et al. [10] study unforeseen change in sensor data signaling malfunctioning or malicious altering. They utilize a neural network prediction function to measure the discrepancy between sensor outputs and the known model of normality. It is clear that the domain-specific changes in the above two studies are different from the change in ours. Sharifzadeh et al. [11] use wavelet coefficients to capture discontinuities of any degree in data and they consider the notion of degree of change as the degree of the changing derivatives at the change point. The definition of change in our problem involves two points. The concept of discontinuity does not apply; a legitimate drop or jump can happen on smooth curves.

The most similar work to our search problem is [2], where a timebox is used to specify constraints on time and values of time series data. A constraint is like (12:00, 13:00, 1, 5) defining a box with an absolute time range from 12:00 to 13:00 and an absolute value range from 1 to 5. Any series with a sub-series' time stamps from 12:00 to 13:00 with values within the range 1 to 5 is returned as query results. In our problem, none of these two kinds of ranges is specified in users' search. This makes the problem much harder and requires a careful treatment of feature compression: A single search in our problem can be corresponding to a number of timeboxes' time ranges and value ranges, which are difficult for users to specify if not impossible. The extension of a timebox in [2] replaces the value range with an angle envelop (ϕ_{min}, ϕ_{max}) where $-\frac{\pi}{2} \leq \phi_{min} \leq \frac{\pi}{2}$ and $-\frac{\pi}{2} \leq \phi_{max} \leq \frac{\pi}{2}$. This new constraint requires slopes of all segments (by connecting two consecutive points) whose time stamps are in an absolute time range to be in the range (ϕ_{min}, ϕ_{max}). This constraint is unable to capture a drop in our search since a legitimate drop can involve data segments with arbitrary slopes in the middle.

There are many alternatives for approximation of time series such as splines, high order polynomials, and wavelets. But it is very difficult to define our problem whose potential solutions involving arbitrary combination of time spans and drops in any of the above transformed spaces. The piecewise linear approximation is naturally suggested by the problem

and leads to intuitive and effective feature representation.

8. CONCLUSION

In this paper, we study the problem of searching for drops in sensor data. The problem is motivated by a real-world situation where users have no idea about the shape and absolute magnitude of data they are looking for but instead they specify their search by certain threshold conditions on relative change in values. The exhaustive search for this problem consumes too much space, which considerably slows down responses. Indexes in the exhaustive search are expensive and cannot improve its performance. In the proposed framework, we design a novel feature space that visualizes target events and the search conditions, invent parallelogram feature representation which is capable of substantially compressing features needed for search, identify the necessary corner points of a parallelogram to support the mapping from search to standard database range queries, and prove the guarantee that no true events are missed in returned results and any false positive returned is within a user-specified error tolerance. Extensive experimental results demonstrate the efficiency of the framework with respect to feature size and search time.

ACKNOWLEDGMENT

9. REFERENCES

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

[2] H. Hochheiser and B. Shneiderman. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. *Information Visualization*, 3(1):1–18, 2004.

[3] E. Keogh. A decade of progress in indexing and mining large time series databases. In *Proc. of VLDB*, pages 1268–1268, 2006.

[4] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. of SIGMOD*, pages 188–228, 2001.

[5] E. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In *Proc. of ICDM*, pages 289–296, 2001.

[6] E. Keogh and M. J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *Proc. of KDD*, pages 239–243, 1998.

[7] Q. Li, I. F. V. Lopez, and B. Moon. Skyline index for time series data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(6):669–684, 2004.

[8] J. Ma and S. Perkins. Online novelty detection on temporal sequences. In *Proc. of KDD*, pages 613–618, New York, NY, USA, 2003.

[9] Y. Qu, C. Wang, and X. S. Wang. Supporting fast search in time series for movement patterns in multiple scales. In *Proc. of CIKM*, pages 251–258, 1998.

[10] L. Reznik, G. V. Pless, and T. A. Karim. Signal change detection in sensor networks with artificial neural network structure. In *Proc. of IEEE CIHSPS*, pages 44–51, 2005.

[11] M. Sharifzadeh, F. Azmoodeh, and C. Shahabi. Change detection in time series data using wavelet footprints. In *Proc. of SSTD*, pages 127–144, 2005.

[12] H. Shatkay and S. B. Zdonik. Approximate queries and representations for large data sequences. In *Proc. of ICDE*, pages 536–545, 1996.

[13] C. Wang and X. S. Wang. Supporting content-based searches on time series via approximation. In *Proc. of SSDBM*, pages 69–81, 2000.

[14] H. Wu, B. Salzberg, and D. Zhang. Online event-driven subsequence matching over financial data streams. In *Proc. of SIGMOD*, pages 23–34, 2004.

[15] Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. In *Proc. of KDD*, pages 336–345, 2003.
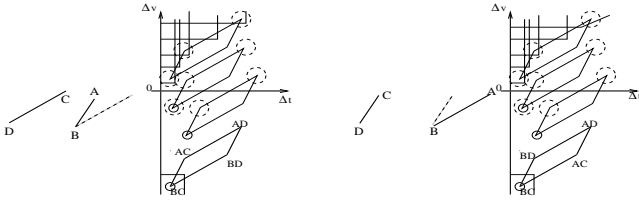
APPENDIX
Corner cases

Figure 25: Boundary conditions of case 2

Figure 26: Boundary conditions of case 3

Case 2 $k_{CD} \geq 0$ and $k_{AB} \geq k_{CD}$. Figure 25 shows this case.

Drop Feature point BC is the (degenerated) lower left boundary. If any drop event occurs in the query region, the feature associated with BC must be in the region. So it is sufficient to record the feature associated with this point. The solid circles in Figure 25 mark this point for each parallelogram.

Jump I As shown in Figure 25, feature segments (BC, AC) and (AC, AD) are the upper left boundary. There are at most five sub-cases for a query region to intersect a parallelogram when AC denotes a jump, which is shown by top two parallelograms in the figure: Feature point BC falls into the region; AC falls into the region; AD falls into the region; part of feature segment (BC, AC) is inside the region but neither feature point BC nor feature point AC is in the region; part of (AC, AD) inside the region but neither AC nor AD is in the region.

Jump II In the case of the second-to-last bottom parallelogram where feature point AC denotes a drop and feature point AD denotes a jump in Figure 25, feature segment (AC, AD) is the upper left boundary. If a query region of jump search intersects a parallelogram in this case, one of the following two sub-cases must be true: feature point AD is in the region; part of feature segment (AC, AD) is in the region but feature point AD is not.

Features to be collected If $\Delta v_{BC} - \epsilon \leq 0$, the feature $(\Delta t_{BC}, \Delta v_{BC} - \epsilon)$ is collected; If $\Delta v_{AC} + \epsilon \geq 0$, the features $(\Delta t_{BC}, \Delta v_{BC} + \epsilon)$, $(\Delta t_{AC}, \Delta v_{AC} + \epsilon)$ and $(\Delta t_{AD}, \Delta v_{AD} + \epsilon)$ are collected; if $\Delta v_{AC} + \epsilon < 0$ and $\Delta v_{AD} + \epsilon > 0$, $(\Delta t_{AC},$

$\Delta v_{AC} + \epsilon)$ and $(\Delta t_{AD}, \Delta v_{AD} + \epsilon)$ are collected.

Case 3 $k_{CD} \geq 0$ and $0 < k_{AB} < k_{CD}$. Figure 26 shows this case. It is the same as case 2 except that AC and BD exchange their positions in feature space.

Features to be collected The conditions and the corresponding features from case 2 apply here with changing Δt_{AC} to Δt_{BD} and changing Δv_{AC} to Δv_{BD}.

The first three cases consider $k_{CD} \geq 0$. The next three cases consider $k_{CD} < 0$. The descriptions are similar. It is sufficient to understand these three cases with the notational convention that broken-line circles label upper left boundary corner points for jump search and solid ones mark lower left boundary corner points. Case 4 is corresponding to case 1. Case 5 is corresponding to case 2. Case 6 is corresponding to case 3.
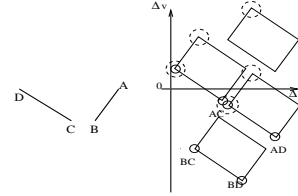

Figure 27: Boundary conditions of case 4

Case 4 $k_{CD} < 0$ and $k_{AB} \geq 0$. Figure 27 shows this case.

Drop Feature segment (BC, BD) is the lower left boundary for drop search.

Jump Feature segment (BC, AC) is the upper left boundary for jump search.

Features to be collected If $\Delta v_{BD} - \epsilon \leq 0$, the features $(\Delta t_{BC}, \Delta v_{BC} - \epsilon)$ and $(\Delta t_{BD}, \Delta v_{BD} - \epsilon)$ are collected; if $\Delta v_{AC} + \epsilon > 0$, the features $(\Delta t_{BC}, \Delta v_{BC} + \epsilon)$ and $(\Delta t_{AC}, \Delta v_{AC} + \epsilon)$ are collected.
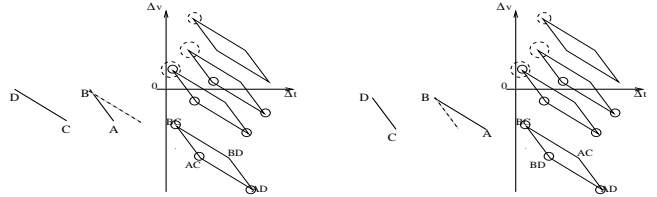

Figure 28: Boundary conditions of case 5

Figure 29: Boundary conditions of case 6

Case 5 $k_{CD} < 0$ and $k_{AB} \leq k_{CD}$. Figure 28 shows this case.

Drop I When AC denotes a drop, feature segments (BC, AC) and (AC, AD) are the lower left boundary.

Drop II When AC denotes a jump and AD denotes a drop, feature segment (AC, AD) is the lower left boundary.

Jump Feature point BC is the (degenerated) upper left boundary.

Features to be collected if $\Delta v_{AC} - \epsilon \leq 0$, the features $(\Delta t_{BC}, \Delta v_{BC} - \epsilon)$, $(\Delta t_{AC}, \Delta v_{AC} - \epsilon)$ and $(\Delta t_{AD}, \Delta v_{AD} - \epsilon)$ are collected; if $\Delta v_{AC} - \epsilon > 0$ and $\Delta v_{AD} - \epsilon \leq 0$, the features $(\Delta t_{AC}, \Delta v_{AC} - \epsilon)$ and $(\Delta t_{AD}, \Delta v_{AD} - \epsilon)$ are collected; if $\Delta v_{BC} + \epsilon > 0$, the feature $(\Delta t_{BC}, \Delta v_{BC} + \epsilon)$ is collected.

Case 6 $k_{CD} < 0$ and $k_{CD} < k_{AB} < 0$. As shown in Figure 29, this case is the same as case 5 except that AC and BD exchange their positions.

Features to be collected The conditions and the corresponding features from case 5 apply here with changing Δt_{AC} to Δt_{BD} and changing Δv_{AC} to Δv_{BD}.