Persistence Based Online Signal and Trajectory Simplification for Mobile Devices

Persistence Based Online Signal and Trajectory Simplification for Mobile Devices

Panagiota Katsikouli
Dept. of Informatics
University of Edinburgh
Edinburgh, United Kingdom
p.katsikouli@sms.ed.ac.uk

Rik Sarkar
Dept. of Informatics
University of Edinburgh
Edinburgh, United Kingdom
rsarkar@inf.ed.ac.uk

Jie Gao
Dept. of Computer Science
Stony Brook University
Stony Brook, NY, USA.
jgao@cs.stonybrook.edu

ABSTRACT

We describe an online algorithm to simplify large volumes of location and sensor data on the source mobile device, by eliminating redundant data points and saving important ones. Our approach is to use topological persistence to identify large scale *sharp features* of a data stream.

We show that for one-dimensional data streams such as trajectories, simplification based on topologically persistent features can be maintained online, such that each new data-point is processed in $O(1)$ time. Our method extends to multi-resolution simplifications, where it identifies larger scale features that represent more important elements of data, and naturally eliminates noise and small deviations. The multi-resolution simplification is also maintained online in real time, at cost of $O(1)$ per input point. Therefore it is lightweight and suitable for use in embedded sensors and mobile phones. The method can be applied to more general data streams such as sensor data to produce similar simplifications. Our experiments on real data show that this approach when applied to the curvature function of trajectory or sensor data produces compact simplifications with low approximation errors comparable to existing offline methods.

Categories and Subject Descriptors

F.2.2 [**Nonnumerical Algorithms and Problems**]: Geometrical problems and computations

General Terms

Algorithms

Keywords

Mobile sensing, spatial data, geometry, persistence, trajectory simplification.

1. INTRODUCTION

Sensing on mobile devices offers opportunities to understand people and environment in great detail: from understanding movement patterns of masses [14] to detecting the context and activities [30, 28] of individuals. However, continuous sensing of locations and physical quantities can produce large quantities of data [4], that can be costly to store, transmit and process.

In this paper, we develop methods to *simplify* sensor data by discarding redundant information and saving only the significant data points. Our goal is to preserve the *sharp* features that represent the important characteristics of the sensed quantity. Figure 1 shows an example where the light levels detected by a sensor on a phone drops and later rises again. Simplification using our method (shown in red) preserves the sharp drop and rise in light levels that is the characteristic of this graph. Many sensor signals show such piecewise constant or linear behavior, where sharp changes signify critical events [34, 28]. In case of Fig. 1, the changes signify the user moving from outdoors to indoors and then outdoors again. The noisy, jittery nature of real data makes it challenging to detect sharp features, and traditional methods (such as low pass filters) that look for smooth features are unable to detect the sharp changes and events.

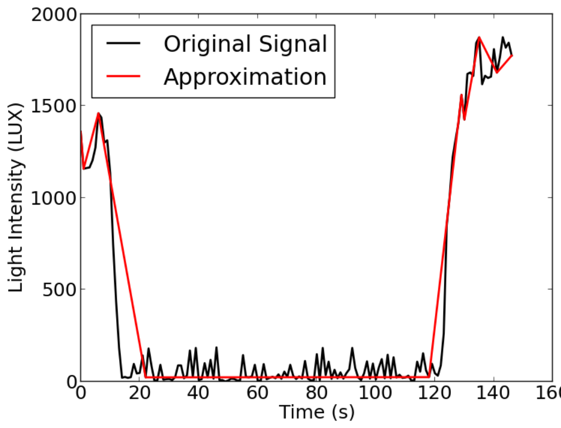

Figure 1. Light levels at 147 data points in Lux (shown in black) recorded by light sensor on a Nexus 5 phone and a simplification with only 10 data points obtained by our algorithm (in red).

Locations and trajectory are important data sensed by mobile devices. GPS receivers give discrete samples of locations producing a piecewise linear trajectory. This trajectory is not smooth and the turns are its most important features. Road intersections, for example, are implied by turns and are thus critical in automated road map generation and refinement [17, 32]. Figure 2 shows a typical GPS trajectory and as can be seen easily, the most important information of this trajectory lies in the small number of turns. Simplifying the trajectory using the sharp features naturally produces compact high quality approximations that are useful in analytics. Due to the importance of trajectory data, recent works have focused on analyzing trajectories through clustering [29, 5], segmenting [35, 3] and find-

.

ing medians [6]). The complexity of these methods increase with the input size, thus simplified inputs open the possibility of analyzing larger datasets and performing analytics on mobile devices.

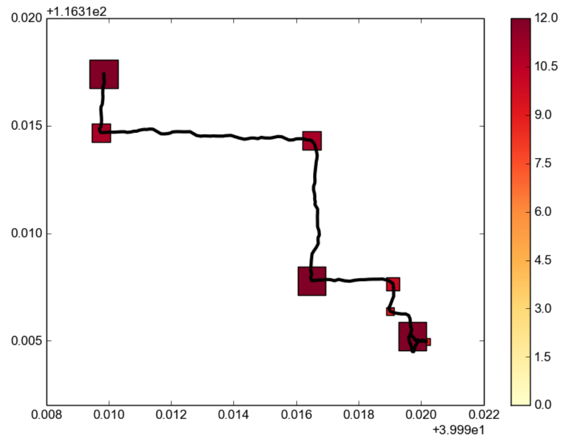

Figure 2. A GPS trajectory consisting of 204 points and a total length of 2.4km. The rectangular markers show the 8 most significant points along the trajectory detected by our method; the size and color of the rectangle grows proportionally to the importance of the turn (the scale to which it survives; this will be explained later). The axes of the plot correspond to the latitudes and longitudes of the points of the trajectory.

Problem description and our contribution. Our goal is to simplify trajectories and signals on low power, battery operated mobile phones and sensors, so that simplified data can be used or transmitted on demand.

Such simplification methods must be efficient and process continuous data streams on the fly, ideally with $O(1)$ time on each incoming vertex. A major challenge is created by the noisy nature of real data (Fig. 1), that can introduce sharp turns, in a local scale resembling the true sharp features we would like to capture. One way to handle noise and unimportant detours is to interpret the data at larger scales that are not affected by noise. Thus we need an algorithm for detecting *persistent* (large scale) features with multi-resolution simplifications.

The main contribution of our method is a simple and online algorithm for topological persistence. Important geometric features of a shape can be seen as persistent features of its curvature function [9] and can be applied to real point clouds through a tangent approximation method [11]. However, these existing methods aim to find smooth features in higher dimensional data and are designed with offline computation in mind. They handle noise by smoothing the data, which may distort the sharp features. Our algorithm in contrast explicitly seeks sharp features and is designed for online embedded computation.

We first describe an $O(n)$ algorithm for computing persistence on one-dimensional data, that is lightweight and suitable for embedded devices. Then we show that in the case of one-dimensional streaming data, such as a time series or trajectory, persistence can be applied online, if the persistence threshold is known. Each incoming data point can be processed and included into the existing simplification or discarded as redundant, in constant time. Analogously, multiple simplification segments can be merged together in time proportional to the number of segments. This implies easy distributed operation, for example, when segments of a trajectory are recorded by different sensors.

The multi-resolution simplification works as follows: The level j simplification has successive vertices at least distance 2^j apart; the level $j+1$ simplification is computed from level j:

1. On the level j simplification, execute persistence to remove redundant vertices and detect important vertices.
2. If successive important vertices are within distance of 2^{j+1}, then sample them suitably to ensure that successive vertices are at least distance 2^{j+1} apart.

We show that the multi-resolution simplification can be executed online, and also works at average space and time complexity of $O(1)$ per vertex. We also show quality bounds on the resulted simplification.

We conduct experiments on real data, including the Geolife [33] trajectory set, and light and cellular signal strength data collected in our own experiments. We find that our online method performs comparably in quality to offline methods and produces approximations with median error of only a few meters using less than 5% of the input vertices.

In the following we first survey related work on shape/curve simplification. Then we introduce the background of persistence idea and discrete curvature. Our algorithm is explained afterwards with experimental results reported at the end.

2. RELATED WORKS

Simplification of curves has been a topic of study in computational geometry, data analysis and related fields. In computational geometry, the problem is typically formulated as finding a subset of vertices from the input vertices of a polygonal curve such that the resulting curve is within ε distance from the original one [15]. The distance of two curves can be measured as Hausdorff distance or Fréchet distance. The most well known algorithm is possibly the Douglas-Peucker algorithm [12], initially suggested in cartography. It starts from a segment connecting the beginning and endpoint of the curve and recursively selects and inserts points furthest away from the current approximation, until the distance is below a user-specified tolerance. Work along the theoretical direction mainly aim for near linear running time algorithms with given error guarantee (see [2] and references therein). Variants of the Douglas-Peucker algorithm can be applied recursively for different resolutions [20, 23, 10]. These papers aim to produce aggressive compression with small Hausdorff errors.

Most of the algorithms in the literature are offline algorithms, assuming that the curves are entirely available. A few of them consider the *online setting*, the same as in our paper, making them appropriate for compressing GPS traces. Furthermore, for the case of simplification of GPS trajectories some applications also need auxiliary information such as time and speed, and accordingly some methods attempt to preserve these quantities [24, 27, 25]. Thus, the Hausdorff distance measure as used for the Douglas-Peucker algorithm is often replaced by a time-aware measure called the Synchronous Euclidean Distance (SED) [24].

The most trivial online compression method for GPS trajectory simplification is Uniform Sampling which simply takes every ith point in the trajectory. This may produce a curve with high information loss. Dead Reckoning is a technique commonly used in calculating one's current position based on previous positions to achieve adaptive sampling [8, 31]. A new sample is kept if it deviates from the prediction significantly. Muckell *et al.* [25] improved over that and proposed a system called SQUISH in which local optimization is used to permanently prune redundant or insignificant points from the original GPS trajectory, while [26, 19] attempt to save energy costs through prediction. Yet another class of algorithms called *direction preserving*, tries to preserve the directional information in the input [22]. From a theoretical angle, Mark de Berg et al. [1] introduce online algorithms using the Hausdorff and Fréchet error functions and show theoretical results about the monotonicity of those error functions under different types of curves. Other works approach the problem from the perspective of moving object databases [31, 21]. Lange et al. proposed a family of tracking protocols [21]. An experimental comparison of some existing online algorithms can be found in [16].

Path segmentation is the problem of subdividing a trajectory with respect to a signal into fewest segments such that the signal function satisfies some property in each segment (consider for example segmenting a path based on cell signal strength). While this question bears some similarity to our aims, existing works [7, 35, 3] take an optimization approach that can only be used offline.

We also note that path compression is often considered together with map matching. The problem is then formulated as trajectory compression with network constraints [18, 8]. In our setting, the significant turns are likely also significant elements in the underlying map although all information is inferred purely from the trajectories, requiring no knowledge of the map data.

3. BACKGROUND

Our online simplification method works on the basis of persistent topology. In this section, we introduce the basic concepts in persistence and discrete curvatures, and describe how persistence helps in path simplification.

3.1 Persistence

Topological persistence, as described in [13], is a technique for simplifying geometric data using topological features of an object. We explain it using the graph of a function f in Figure 3. Among the ups and downs of $f(x)$, the hill and the valley at d and a are intuitively *large* features that should be preserved in a simplification of $f(x)$. The small bump and depression created by fluctuation around points b and c are clearly less significant.

Persistence sweeps the values of the function f from minimum to maximum, and keeps track of the connected components of sublevel sets. When the current sweep value is y, we update the connected components of the domain x, that satisfy $f(x) \leq y$. Connected components are created (or born) when the sweep reaches minima such as a and c. These components merge when the sweep reaches maxima like b – when the smaller (more recent) component is merged into the larger (and considered to have died).

Persistence produces (min, max) pairs corresponding to birth and death of components, that give rise to a set of intervals or bars, often called a *barcode* (Fig. 3(c)). The smaller bars in the barcode correspond to less important characteristics of the function. Removing them can produce a simplification, which still preserves the function at the important points which are the end-points of the larger bars.

3.1.1 *Use of persistence in path simplification*

Persistence lets us explicitly identify and remove noise and less important features, while preserving the important points (such as a and d), thus locations of large turns in trajectory, or exact time a sensor signal jumps, are preserved exactly. However, persistence cannot be applied directly to the graph of a trajectory the way we applied it to the function $f(x)$, since the maxima and minima with respect to a given direction may not always correspond to the significant points. For example, see Fig 4(a). The path from s to t has the most important turns at a, b, c and d. But applying persistence using the coordinates shown here detects only s, t and z as these are the only maxima and minima. A change of coordinates maybe able to detect some of the important points, but not all. We would like a way to detect a, b, c, d irrespective of coordinates. Observe that such a path with only a, b, c, d as intermediate points (shown in dashed grey) is a good approximation of the original path. This problem can be solved by applying persistence to the curvature function of an object instead of its raw coordinates. However, existing approaches as in [11] are designed for offline computation and rely on smoothing the data, which makes them unsuitable for our purpose.

3.2 Discrete Curvature

We thus look at curvature as a function to analyze our input path – higher curvatures signify sharper turns and therefore can be considered representative of important features. Curvature is also a *complete descriptor* of a curve, that is, it encodes all the information needed to reconstruct the curve. Our inputs are sequences of locations, which constitutes discrete data, such as a piecewise linear path. In this case, curvature can be measured as the counterclockwise angle of turns (Fig. 4(b)). Clockwise turns are measured as negative curvature.

We can intuitively expect significant points along the trajectory to have high absolute discrete curvatures. For example in Fig. 4(a), removal of z and similar vertices yield approximations close to the original curve as shown by the dashed line.

4. ALGORITHM

In this section, we first describe an efficient $O(n)$ algorithm for persistence. Using the fact that the data is one-dimensional and arrives sequentially in time, we then develop the distributed and online versions. Finally we will describe how this algorithm naturally extends to a multiscale format.

We assume our path is given as a sequence of locations $P = \{p_i\}$. We use the shorthand notation P_{ij} to represent the entire trajectory between p_i and p_j. We can assume without loss of generality that no two vertices have exactly the same value (we can resolve ties arbitrarily, for example, by IDs). We aim to compute the β-persistent vertices of the function f. That is, the (min, max) pairs from persistence that differ in their f-value by more than β.

4.1 $O(n)$ Persistence Algorithm

To make the paper self-contained, we present how to implement the persistent simplification algorithm in a centralized offline setting at $O(n)$ cost suitable for embedded devices, with adaptations that make it easier to extend to distributed and online cases. The main contribution is a distributed online algorithm described in the following subsection.

Phase 1. We first traverse the path and identify all local maxima and minima of f along P. Then at each minimum vertex $m_i = p_i$ (we use m_i to represent the fact that vertex p_i is a minimum), we start constructing a connected component $C(p_i)$ (interchangeably written as C_i) by appending neighboring vertices to both ends. The component can be maintained simply as a pair of end-points that demarcates it in P. Any connected component along a path has exactly 2 *neighboring* vertices – let us call these q_1 and q_2 – with one on each side. We grow the component by including one of these vertices in each step: if $f(q_1) < f(q_2)$, then we add q_1 to C_i, else we add q_2. We repeat this process until a vertex x, which is a maximum of f is added to C_i, then we stop growing C_i.

At each maximum, we keep track of the components to which it has been added. When a maximum x has been added to 2 components, say C_1, C_2, we merge these into the component that started earlier. That is, if $m_1 > m_2$, then we merge C_1 into C_2, and output a bar (m_1, x) and set $C_2 = C_2 \cup C_1$ (note that in a path, concate-

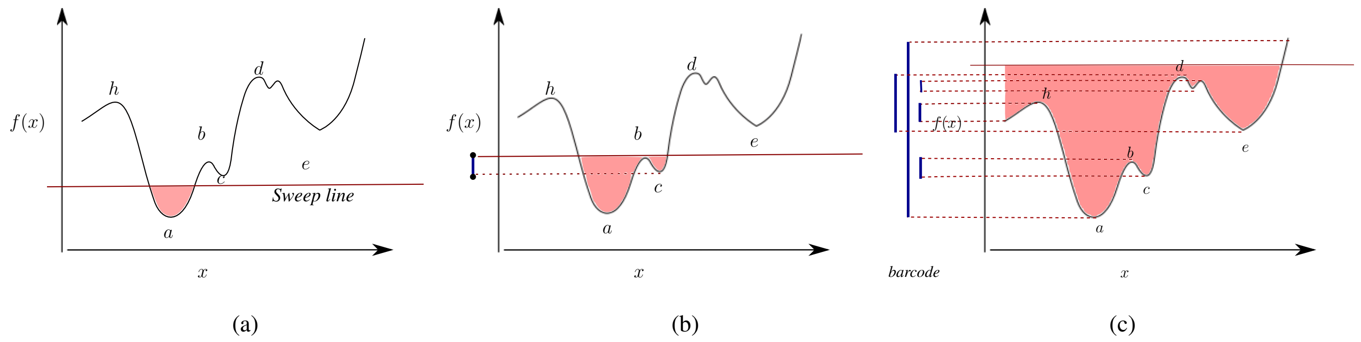

Figure 3. Persistence analysis of the sublevel sets of a part of function $f(x)$. (a) The values of $f(x)$ are swept with a line in increasing order. The first point encountered is the minimum at a and a new component of the sublevel set is created and continues to grow. (b) Next component created at minimum c. These components meet each other when they both reach the maximum at b. Component of c merges into the larger component of a. The blue bar on the left shows its *lifetime*. (c) After the sweep has covered this region, the *barcode* (blue bars on the left), show paired *(birth, death)* of components: (c, b), (e, d), etc giving the lifetime of each feature. The *small bars* such as (c, b) represent small features.

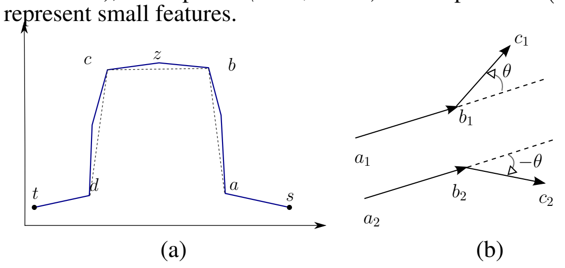

Figure 4. (a) On path from s to t, the important points are a, b, c, d. But persistence in the given coordinates finds z instead of a, b, c, d. Analogously, Douglas-Peucker [12] identifies z first, and misses the importance of high curvature points. (b) Curvature as signed angle of turn.

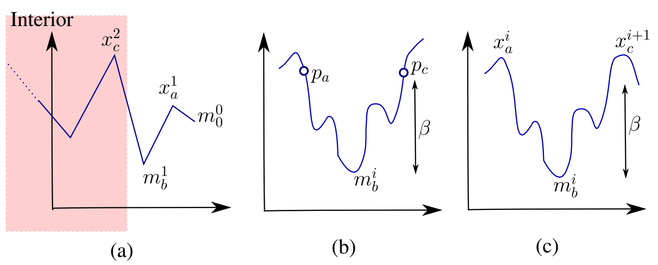

Figure 5. (a) Interior and non-interior vertices in a simplification. (b) Existence of vertices p_a and p_b that differ by β from m_b^i, with m_b^i the smallest min between them, implies that m_b^i must be $\beta-$persistent: Lemma 4.1. (c) If x_a^i, m_b^i and x_c^{i+1} are $\beta-$persistent, then m_b^i must be the smallest min between x_a^i and x_c^{i+1}, and must differ by β from both: Lemma 4.2.

nating two consecutive subpaths is done by just reassigning one of the end-points). After this merge, neither of the end points of C_2 is a maximum, and it is possible to continue growing C_2 as before.

The process of growing and merging components continues until all vertices of P belong to the same component. When the algorithm ends, we have a set of bars corresponding to different segments of the path.

$\beta-$persistent simplification $P(\beta)$. The sequence of vertices that lie at the end of bars of length greater than or equal to β constitute a $\beta-$persistent simplification of P. Let us use $x^0, m^0, x^1, m^1, \ldots$ to represent the interleaved sequence of minima and maxima in the $\beta-$persistent simplification. We use m_j^i to represent that minimum i occurs at vertex p_j, and we use it interchangeably with p_j. Analogous notations hold for maxima. See Fig. 7 for an example of the output of this simplification.

Since the end points of a trajectory are always important, we make a few special selections at the boundaries to keep things consistent. Let us call this Phase 2:

Phase 2. We always select an end point to be part of the simplification. An end point p_0 must always be a maximum or a minimum, since no two vertices have same f value. Let us say it is m_0^0. Suppose that m_b^1 is the first $\beta-$persistent extremum after m_0^0 selected in phase 1. Without loss of generality, we are assuming it is a minimum. Suppose $x_a^1 = \arg\max_{p \in P_{0b}} f(p)$, that is, x_a^1 holds the highest value of f between m_0^0 and m_b^1, then we also include x_a^1 in the simplification. (analogously, when the first $\beta-$persistent vertex is a max, we will select the lowest minimum.)

Interior vertices. We define as *Interior vertices* of a simplification any vertex *other than* the terminal vertices (such as m_0^0), their neighbors (such as x_a^1), and the next vertex selected in phase 1, such as m_b^1. Observe that all interior vertices must be selected in phase 1. We refer to m^0, x^1 and m^1 as *non-interior vertices* (see Fig. 5).

Complexity of the algorithm is $O(n)$. There are at most $O(n)$ maxima and minima. The maximum number of components is therefore $O(n)$. The number of growth updates to components equals the number of vertices, which is $O(n)$ and the number of merges is bounded by the number of maxima, which is $O(n)$. Thus the total complexity of the algorithm is $O(n)$. The argument that this algorithm produces the same result as the standard sweep algorithm described in the previous section is straightforward, and omitted here.

The efficiency of the algorithm makes it suitable for processing large volumes of data, and for use on small devices.

4.1.1 Properties of the $\beta-$persistent simplification.

After removing bars of persistence less than β, we are left with a sequence of vertices that alternate between maxima and minima. Each minimum m is paired with a maximum x such that $f(x) - f(m) \geq \beta$.

We now show a few properties that will be important in constructing the distributed and online versions of our algorithm.

Lemma 4.1. *For any minimum m_b^i if there exist vertices p_a and p_c such that both the following conditions hold (See Fig. 5(b)):*

1. $m_b^i = \arg\min_{p \in P_{ac}} f(p)$. *That is, m^i is the minimum of f in the interval between p_a and p_c*

2. $f(p_a) \geq f(m^i) + \beta$ *and* $f(p_c) \geq f(m^i) + \beta$

Then, m_i will be selected as a $\beta-$persistent element in the simplification by the phase 1 of persistence algorithm.

PROOF. Suppose to the contrary, that m^i is not β-persistent, implying that the maximum x where the component $C(m_b^i)$ dies, satisfies $f(x) < f(m^i) + \beta$. Without loss of generality, suppose no other vertices on P_{ac} other than p_a and p_c satisfy the conditions.

In that case, there cannot be a maximum $x \in P_{ac} : f(x) \geq f(m_b^i) + \beta$ (other than possibly p_a or p_c). Since then, by the intermediate value theorem, there will be vertex between m_b^i and x satisfying the conditions of the lemma.

Now consider the evolution of $C(m_b^i)$. Since every other minimum in P_{ac} is higher than $f(m^i)$, $C(m_b^i)$ cannot die at any maximum in P_{ac} (other than possibly p_a ors p_c). Maximum x where $C(m_b^i)$ dies must be at p_a, p_c, or outside P_{ac}. Thus, when $C(m_b^i)$ dies p_a or p_c must be in $C(m_b^i)$, and x must satisfy $f(x) \geq f(m^i) + \beta$, contradicting our hypothesis. □

The next lemma shows a converse result that for a $\beta-$persistent vertex m, its neighboring $\beta-$persistent vertices must differ by β, and m must be the extreme critical point between them.

Lemma 4.2. *For the interleaved sequence of $\beta-$persistent maxima and minima $\ldots x_a^i, m_b^i, x_c^{i+1} \ldots$, where m_b^i is an interior vertex (see Fig. 5(c)), the following properties hold:*

1. $m_b^i = \arg\min_{p \in P_{ac}} f(p)$. *That is, m^i is the smallest minimum of f in the interval between a and c*
2. $f(x^i) \geq f(m^i) + \beta$ *and* $f(x^{i+1}) \geq f(m^i) + \beta$

PROOF. Part 1. If to the contrary, there was another minimum m_d^j in P_{ab} such that $f(m_d^j) < f(m_b^i)$, then components C_b and C_d would meet at some maximum x in P_{bd} and C_d will survive as the unique minimum between x_i and x_{i+1}. This contradicts the hypothesis that C_b is the surviving minimum. (Observe that if the maximum x existed, it must satisfy $f(x) < f(m^i) + \beta$, since other wise by Lemma 4.1, x must be in the simplification and $\ldots x_a^i, m_b^i, x_c^{i+1} \ldots$ cannot be a sequence in the simplification.)

Proof of Part 2. Suppose w.l.o.g $f(x^i) < f(x^{i+1})$. And for the sake of contradiction, assume $f(x^i) < f(m^i) + \beta$. Now suppose that during the run of the algorithm, the component $C(m^i)$ meets component $C(m^j)$ at x^i. If $f(m^j) < f(m^i)$, then $C(m^i)$ merges into $C(m^j)$, and m^i cannot be in a β-persistent simplification.

Alternatively, if $f(m^j) > f(m^i)$, then $f(x^i) < f(m^j) + \beta$, and therefore, x^i cannot be in a β-persistent simplification. □

Observe that the lemma above is unrelated to m^i being paired with either of its neighboring maxima x^i and x^{i+1}. It establishes the important fact that the persistent maxima neighboring a persistent minimum are separated from it by at least β irrespective of the pairing of the minimum.

We proved the properties only for the minima of f; corresponding properties hold for the maxima.

4.2 Distributed and online algorithms

The computation of $\beta-$persistent simplifications can be modified to distributed and online algorithms respectively. We first make a few observations about the output – that having computed the simplification on a finite path, extending the path at either end cannot affect the $\beta-$persistent simplification of the interior vertices.

Observation 4.3. *An interior vertex of the simplification can not be non-persistent in an extended path.*

This follows from Lemma 4.1 using the fact that f does not change at an interior vertex or its neighboring vertices.

Observation 4.4. *A vertex not selected in either phase 1 or phase 2 cannot become persistent on addition of more vertices at either end.*

This follows from Lemma 4.2 using the fact that f does not change on existing path P and interior vertices, and the terminal point is part of $P(\beta)$ and the smallest/largest critical point (say x) between the terminal point and the first interior point are also part of $P(\beta)$ from phase 2. That is, any vertex rejected in the interior portion cannot be important in the larger path. Using Lemma 4.2, any vertex in the non-interior portion that may be persistent must be the lowest minimum or the highest maximum, and both these have been selected in phase 2. Thus, it suffices to consider only the vertices in the simplification of the existing segment to extend it.

Distributed algorithm. This algorithm assumes that the path has been segmented into multiple pieces, and the simplification is known for each piece. We assume that each segment is long enough to have interior vertices. Our goal is to merge these simplifications into a single simplification, in time proportional to the number of pieces. This has applications where a device periodically uploads simplified data to a server, or when different parts of a trajectory are recorded by different devices.

On any segment, the relevant portion is a sequence of $P(\beta)$ given by q^0, q^1, q^2, q^3. Where:

- q^0, q^1 and q^2 are the non-interior vertices, with q^0 being the terminal vertex;
- q^3 is the interior vertex that is neighbor to q^2.

Observations 4.4 and 4.3 suggest that we can restrict our attention entirely to the non-interior vertices of the simplifications to merge them. The rest of the simplified segments can be simply concatenated.

Thus, in merging two segments sharing an end-point, we need to take only a sequence of 7 vertices: $q^{-3}, q^{-2}, q^{-1}, q^0, q^1, q^2, q^3$ covering the non-interior vertices of two segments (see Fig. 6(a)). Now, any vertex among these that is $\beta-$persistent must also be an interior vertex in the combined simplification. Thus we then run phase 1 of the algorithm on this sequence to obtain the $\beta-$persistent vertices here.

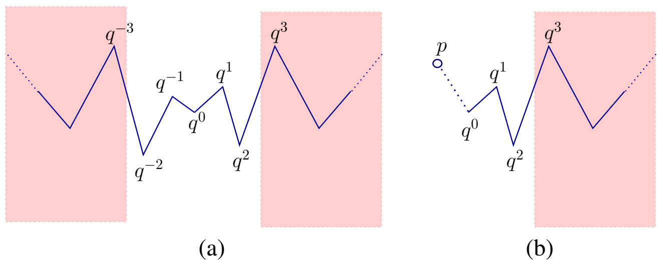

Figure 6. (a) Concatenating two precomputed simplifications. q^0 is the common vertex. Persistence decision can change only for non-interior vertices, we can make the decision by executing persistence on the segment q^{-3} to q^3. (b) Online algorithm: p is incoming new vertex. We are trying to decide if q^2 is interior.

Online algorithm. We would like an online method, where each vertex is processed and incorporated into the simplification immediately on arrival. We maintain a working list q^3, q^2, q^1, q^0 as before. Remember that in this list, q^3 is interior, q^2 was selected in phase 1, but is not interior. Thus q^2 is the first candidate to become interior – see Fig. 6(b) – and that is the decision we are trying to make. This will be the case, for example (by Lemma 4.1), if on both sides of q^2 we can find vertices that are higher than q^2 by β in f value, and q^2 is the lowest minimum between them.

Now suppose a new vertex p arrives. Without loss of generality, let us assume that q^2 is a minimum.

1. If q^0 becomes a maximum and greater than q^1, set $q^1 := q^0$.
2. If q^0 becomes a minimum and $f(q^0) < f(q^2)$, then $q^2 := q^0$, and $q^1 := p$.
3. If $f(p) > f(q^2)$ and $f(p) \geq f(q^2) + \beta$, assign $q^3 := q^2$; and set $q^2 := q^1 := p$.
4. $q^0 := p$

Step 1 simply updates q^1 to be the highest non-interior maximum. Step 2 updates q^2 to be the lowest non-interior minimum. Step 3 takes the important action – if there is evidence that there are vertices $\beta-$separated on either side of q^2, then we promote q^2 to be interior, and proceed to mark list vertices arrived after q^2 as non-interior. The correctness of this step is implied by Lemma 4.1. The final step simply updates q^0 to be the current terminal vertex.

Theorem 4.5. *The online and distributed algorithms produce the same $\beta-$persistent simplification as the centralized algorithm.*

The proof follows using Lemmas 4.1 and 4.2. We omit the proof here due to lack of space.

Complexity. It is easy to see that both the distributed and the online versions run in $O(n)$ time for a path of length n. In the distributed case, the merging of two segments occurs in constant time, since it needs to compute persistent points on a constant sized input. In the online algorithm, each new vertex is incorporated at $O(1)$ cost.

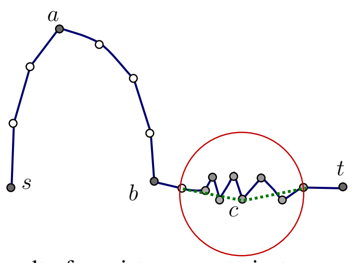

Figure 7. Result of persistence on trajectory curvature. Selected vertices are shaded. Vertices on segment ab do not get selected, in spite of having high curvature. Vertices such as c get selected due to noise. Taking a larger neighborhood (red circle) shows c has small curvature at larger scale.

Outputs of persistence algorithm. Figure 7 shows the result of the persistence algorithm on the curvature of a trajectory. Observe that persistence selects a and b because their curvatures differ by more than β, while vertices in between, that individually have curvatures higher than β, simply form a curved line without any significant sharp feature.

Vertex c has high persistence since curvature varies sharply in its neighborhood due to noise. However, if we take a larger neighborhood around c, shown by the red circle, then with respect to the more distant neighbors, c has a small curvature more representative of the general progress of the trajectory through this region. Note that a and b have high curvatures even at the larger scale. In the next section, we address this problem of handling noise.

In our experiments, we find that a threshold of $\beta = 10\,\text{deg}$ produces good results.

4.3 Multi-resolution simplification

The curvature function, and derivatives in general, tend to amplify the noise in the data – creating extreme critical points – that cannot be removed by persistence alone. We therefore suggest a multi-resolution simplification where discrete curvature κ is considered at different scales, to reduce the impact of noise. The method can be applied to any function, we use curvature for ease of explanation and its importance in trajectory and signal compression.

The algorithm works using multiple levels or scales of simplification. At level j, we have a scale $s = 2^j$ simplification P^j, which is defined as: *For any two successive vertices $u, v \in P^j$, $d(u, v) \geq s$, where d is a suitable distance measure.* In the case of a trajectory, d can be the Euclidean distance.

Our algorithm uses the output of level j as the input to level $j+1$. That is, on $\kappa(P^j)$ we apply the persistence algorithm to select $\beta-$persistent vertices, and then sample these vertices to ensure that any pair of successive vertices are 2^{j+1} apart. The output is P^{j+1}, which contains only a subset of P^j.

Observe that for a vertex v occurring in both P^j and P^{j+1}, $\kappa(v)$ can be different at the different levels or scales, since the neighbors of v can be different in the two cases.

Sampling of vertices of P^j to compute P^{j+1} can be done with different strategies. We use a simple heuristic: Scanning P^j from the start, for every pair of successive vertices that are within distance $2s$, remove the one with the smaller absolute curvature and repeat until the sampling satisfies the definition of P^{j+1}. The rationale is that a vertex with smaller absolute curvature represents a shallow and less significant turn.

Observation 4.6 (Linear size and complexity.). *Assuming that the maximum velocity of the moving object is bounded by some constant v, and that location measurements are taken at (approximately) equal intervals, the multi-resolution representation is computed in $O(n)$ time and space, where n the number of vertices in the input.*

This follows simply from the exponential growth of the spacing between vertices with increasing levels, which implies an exponential decay of number of vertices through levels. An alternate way to state it, which does not rely on equal intervals of location sampling will be that for a trajectory recorded over time T and with some bounded velocity, the space and time complexity of processing are both $O(T)$.

Online multi-resolution algorithm. The multi-level $2s$ selection can be implemented in the online version of the algorithm. Here, we can execute the persistence based online selection on P^j to maintain a copy $P^j(\beta)$ of the trajectory. We also maintain P^{j+1}_{temp} as vertices from $P^j(\beta)$ guaranteed to be $2s$ apart. When a new vertex is selected as an interior point of $P^j(\beta)$, we place it as the terminal vertex of P^{j+1}_{temp} and measure its distance to its neighbor in P^{j+1}_{temp}. If the distance is less than $2s$, we remove the one with lower absolute curvature. P^{j+1} consists of all vertices of P^{j+1}_{temp} except the most recent (terminal) vertex. Note that we introduce lists $P^j(\beta)$ and P^{j+1}_{temp} for ease of explanation. These are not required in an actual implementation.

Discussion. An example of the multi-resolution algorithm in action can be seen in Fig. 8. Here Fig. 8(a) shows that the large scale sharper turns are preserved to higher scale approximations, while noise and small turns are eliminated quickly. Such deviations can also be caused by erroneous data, for example, GPS errors, and are not easy to distinguish from true movements of the user. In our

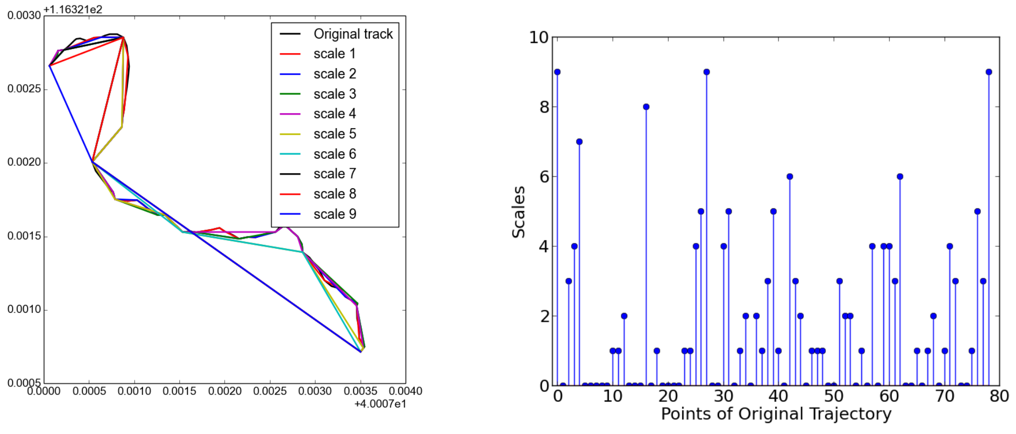

Figure 8. A GPS trajectory consisting of 79 points, and approximations at different scales. (a) Scale 2^3, 2^4, 2^5 and 2^6 metres approximations have 25, 18, 11 and 7 points respectively. (b) Shows the different levels/scales to which vertices survive. Survival scales (or persistence) of vertices produces a natural ranking.

experiments we observed that those features are usually small and do not survive to the higher scales of the multi-resolution algorithm.

Figure 8(b) shows the scales to which different vertices survive. This can be seen as another layer of persistence, where all elements start at the same time, but some survive longer (to higher scales). The more *persistent* features reveal the more important properties of the data. This ranking of vertices is particularly useful in creating adaptive resolution reconstructions, by considering more important vertices first.

Constant complexity per vertex. The online algorithm processes each incoming vertex in constant time on average. This follows again simply from the exponential decay of number of vertices through levels and thus that a vertex on average only “rises” to a constant level in the hierarchy.

5. IMPLEMENTATION AND EXPERIMENTS

We implemented our algorithm and ran experiments on two different datasets: (i) GPS trajectories near Beijing from Microsoft’s GeoLife database [33] and (ii) Sensor signal datasets recorded on a Google Nexus 5 smartphone in Edinburgh, UK.

Experiments show that our algorithm produces efficient and effective simplifications. In particular:

- Our online algorithm generates compact approximations to trajectories with low error comparable to offline methods.
- On sensor signal data, such as light level and cell signal strength, it performs better than Fourier transform based compression.
- Detects the significant turns in trajectories with high accuracy; removes noise and short detours to produce intuitive results.

Choice of persistence threshold β. In these experiments, we used a persistence threshold of $\beta = 10\,\text{deg}$. Intuitively, β should not be too small, since that will preserve very small changes in curvature that are not significant. On the other hand, β should not be too large, since that may miss important features. By conducting some trials with different values of β, we found $\beta = 10\,\text{deg}$ gives more intuitive results than others. However, a thorough study of the impact of the value of β remains to be done in future.

5.1 Existing methods

We compared our algorithm to several existing methods. We briefly describe them here.

Douglas-Peucker [12] is a heuristic that works in rounds and takes as a parameter the maximum allowed error of approximation, ϵ. It starts with a line segment connecting the endpoints of the path P as the simplest approximation. At each round, it adds to the approximation the vertex on P that is furthest from the current approximation. The rounds continue until all vertices of the trajectory are within a distance ϵ of the approximation.

This method has a natural multi-resolution property, since points added earlier are more important and have a greater impact on the error. However, the method does not naturally extend to online versions. The worst case running time of the method is $O(n^2)$, although in practice it runs much faster.

Iri-Imai [15] This algorithm is more resource intensive but produces optimal results. It takes as a parameter the maximum error ϵ, and constructs a graph G on the set of vertices, with the property that $(u, v) \in G$ if the straight line (u, v) is within a distance ϵ from P_{uv}. Then it computes the shortest path in G that connects the end-points of P. This is naturally the smallest (fewest link) approximation of P that is within an error of ϵ. This algorithm is optimal in the sense of approximation errors. Its drawback is the running time of $\mathcal{O}(n^2)$ to construct G. This strategy is not suitable for online computations.

FFT (Fast Fourier Transform) Fourier transform is a classical method in signal analysis that decomposes signals into constituent sine and cosine waves. Compression can be achieved by preserving the larger low frequency components and removing high frequency noise. We use the discrete cosine transform to achieve this effect. This method is good for preserving smooth properties of signals while eliminating noise, which is not suitable for our goal of preserving the sharper features. The method also does not work in online settings. We compare the accuracy of our method with FFT based compression for the sensor data of light levels and cellular signal strength.

5.2 GPS trajectory simplification

Figure 9 shows the result of simplification using our algorithm on real data on a map and for a couple of hand drawn trajectories.

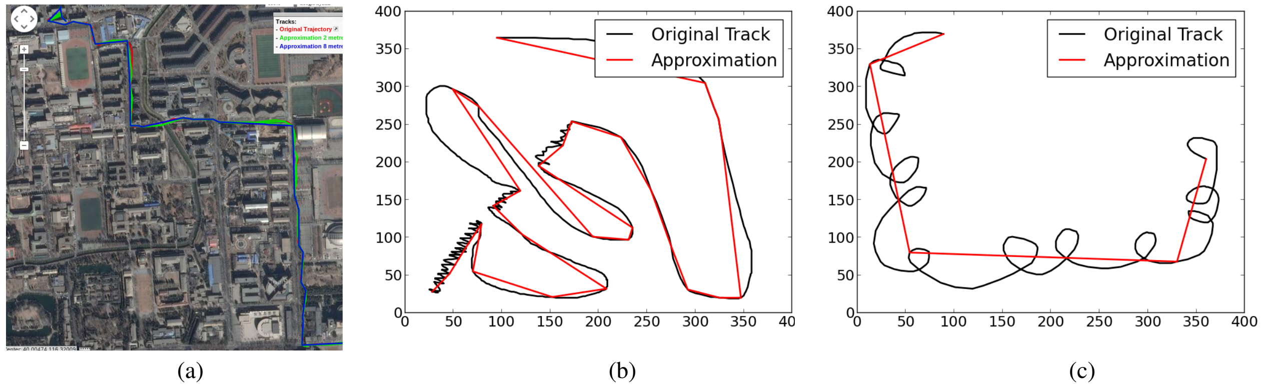

Figure 9. (a) GPS trajectory (red) and its approximations. Green track: scale $s = 2^1$ metres. Blue track scale $s = 2^3$ metres approximation. (b) A hand drawn trajectory of 743 points, approximated by 23 points. (c) A hand drawn loopy curve of 266 points approximated by 5 points.

We tested the performance of our algorithms on a dataset of 500 random GPS Trajectories from the GeoLife database [33]. These trajectories were from mobility inside a city, typical trajectories were a few Km in length. The multiscale persistence algorithm was applied on the discrete curvature function of each trajectory. The distance of a trajectory P from its approximation P' was computed as the Hausdorff distance:

$$d_H(P, P') = \max\{\sup_{p\in P} \inf_{q\in P'} d(p,q), \sup_{q\in P'} \inf_{p\in P} d(p,q)\}$$

with d measured in the L_2 metric. (here we consider both P and P' as the continuous pointsets along the polylines instead of discrete vertices.)

Because the trajectories represent coordinates on the surface of the Earth, we used a modified L_2metric using the Haversine distance as follows:

$$\begin{aligned} a &:= \sin^2(\Delta\phi/2) + \cos\phi_p \cdot \cos\phi_q \cdot \sin^2(\Delta\lambda/2) \\ c &:= 2 \cdot \arctan(\sqrt{a}, \sqrt{(1-a)}) \\ d(p,q) &:= R \cdot c \end{aligned}$$

Where ϕ_p and ϕ_q are latitudes, $\Delta\phi$ is difference in latitudes, and $\Delta\lambda$ is the difference in longitudes. R is mean radius of the earth (6,371km).

We compared the size versus approximation error results of our algorithm on the 500 GPS trajectories against the results of the Douglas-Peucker and the Iri-Imai algorithms. Our algorithm produces a ranking of vertices based on the scale to which they survive (Fig. 8). Douglas-Peucker also produces a natural ranking given by order in which they are added to the simplification.

For Iri-Imai, since the algorithm itself has an $O(n^2)$ complexity which is impractical to run on large trajectories, we divided the trajectories into sequences of 200 vertices, computed the simplifications on these segments, and then concatenated the results. For our algorithm, we used a persistence threshold of $\beta = 10\,\text{deg}$.

The results are shown in Figure 10, for max, mean and median errors. As seen from the plots, Iri-Imai produces the smallest errors. The persistence algorithm, even with an online approach, performs similar to Douglas Peucker. The most important observation is that the median error drops to a few meters with only 5% of the vertices in approximation. Note that GPS localization itself is likely to have a few meters of error.

Execution times. The average running times per trajectory on an Inter(R) Core(TM) i5-3470 CPU at 3.20 GHz with 8GB RAM, running python 2.7 on Scientific Linux 6.4, are shown in Table 1.

Persistence	Douglas-Peucker	Iri-Imai
0.00328	0.0181	5.582

Table 1. Average running time in seconds per trajectory on a set of 500 random Geolife trajectories.

Detection of significant turns. To test that the algorithm detects the important turns at a suitable scale, we created artificial trajectories with clear points of *important turns*. The trajectories follow straight segments of length between 20 and 40 units, and successive segments are at an angle of 75 deg to 105 deg, implying important turns. There were about 100 Vertices along each segment which were perturbed randomly by noise by upto 2 units, which is 10% of the segment length. At a suitable scale, such as scale 10 (half of segment lengths), we expect the simplification to select a vertex near each turn in the artificial trajectory.

Thus we executed the multi-resolution simplification and considered the scale 10 simplification. In Fig. 11 we plot the fraction of real turns that have a level 10 simplification vertex within a radius x. The plot shows that most points of turn are well represented – nearly 90% turn points are represented by a simplification vertex within 2 units – which is the margin of noise.

In Fig. 12 we show the significant turns as detected by our algorithm on a small set of GPS trajectories, with more important vertices shown with larger and darker squares. The busier regions of the domain have a concentration of large scale sharp turns. Which suggests that busy/popular regions, or hotspots, can be detected easily from the simplified data without the need for complete GPS traces.

5.3 Simplification of sensor signal data

We collected the data of cell signal strength and of the light intensity, using a Nexus 5 mobile phone, to test the simplification of timeseries data, while maintaining important features needed for activity and context recognition.

We used the light intensity and the cell signal strength data of 20 different small trips in the city centre. We used the multiscale persistence algorithm on the discrete curvature function of the light intensity and the cell signal strength datasets and used the L_1 metric for the scale s simplification. A graph of light intensity and its simplification can be seen in Fig. 1, while a graph of cell signal strength simplification can be seen in Fig. 13(a). The cell signal strength is shown in ASU[1].

[1]ASU (Arbitrary Strength Unit) is an integer value proportional to

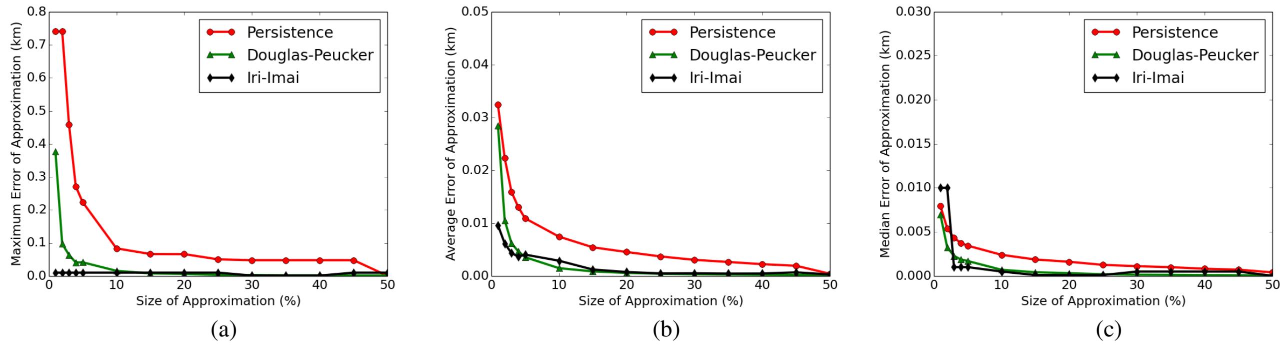

Figure 10. (a) Maximum (b) Average and (c) Median error of approximation per size of approximation. The performance of the persistence algorithm is close to Douglas-Peucker and Iri-Imai.

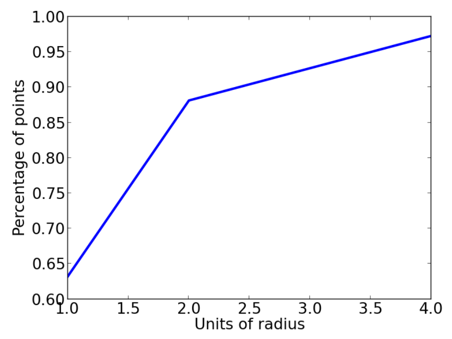

Figure 11. Percentage of points of the approximation close to the significant turns of the curve. Noise was added between points of sharp turns on artificially and randomly generated curves.

In Figures 13(b) and 13(c) we show the median approximation error against the size of simplification for our algorithm, Douglas-Peucker, Iri-Imai and FFT (or DCT). FFT has the worst performance in both cases, since its approach of approximating smooth features does not work well on jittery sensor data. In particular, as seen in Fig. 1, the variation in light signals is in the range of about 1000Lux, thus an error of several hundred Lux is unacceptable for any application.

The performance of our algorithm using online computation is close to Douglas-Peucker and Iri-Imai, that use offline computation with knowledge of entire trajectories.

6. CONCLUSION

Detecting and preserving sharp features accurately is important in sensor data processing. We presented an online algorithm for topological persistence that preserves sharp features. We extended the algorithm to produce adaptive resolution approximations, and showed that it effectively simplifies real trajectory and sensor data. Other applications of this simplification and use in trajectory clustering, segmentation, signal reconstruction etc remains to be investigated. As part of future work, we plan to implement our technique on mobile devices and evaluate its real time operation in data gathering.

Acknowledgements Jie Gao would like to acknowledge support from NSF through CNS-1217823, DMS-1221339, and DMS-1418255.

7. REFERENCES

[1] M. A. Abam, M. de Berg, P. Hachenberger, and A. Zarei. Streaming algorithms for line simplification. In *Proceedings of the Twenty-third Annual Symposium on Computational Geometry*, SCG '07, pages 175–183, New York, NY, USA, 2007. ACM.

[2] P. K. Agarwal, S. Har-Peled, N. H. Mustafa, and Y. Wang. Near-linear time approximation algorithms for curve simplification. In *Proceedings of the 10th Annual European Symposium on Algorithms*, ESA '02, pages 29–41, London, UK, UK, 2002. Springer-Verlag.

[3] B. Aronov, A. Driemel, M. van Kreveld, M. Löffler, and F. Staals. Segmentation of trajectories on non-monotone criteria. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1897–1911. SIAM, 2013.

[4] J. Biagioni, A. Musa, and J. Eriksson. Thrifty tracking: online gps tracking with low data uplink usage. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 486–489. ACM, 2013.

[5] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. *International Journal of Computational Geometry & Applications*, 21(03):253–282, 2011.

[6] K. Buchin, M. Buchin, M. Van Kreveld, M. Löffler, R. I. Silveira, C. Wenk, and L. Wiratma. Median trajectories. *Algorithmica*, 66(3):595–614, 2013.

[7] M. Buchin, A. Driemel, M. van Kreveld, and V. Sacristán. Segmenting trajectories: A framework and algorithms using spatiotemporal criteria. *Journal of Spatial Information Science*, (3):33–63, 2014.

[8] H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *The VLDB Journal*, 15(3):211–228, Sept. 2006.

[9] G. Carlsson, A. Zomorodian, A. Collins, and L. Guibas. Persistence barcodes for shapes. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, pages

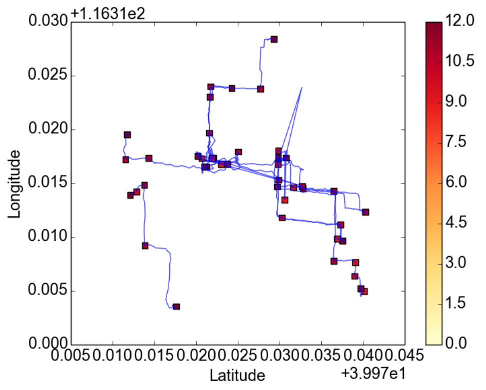

Figure 12. The most significant turns of the trajectories are represented by rectangles of various sizes. More important vertices are marked with larger and darker squares. A concentration of large scale vertices implies popular regions or hotspots.

the received signal strength measured by a mobile phone.

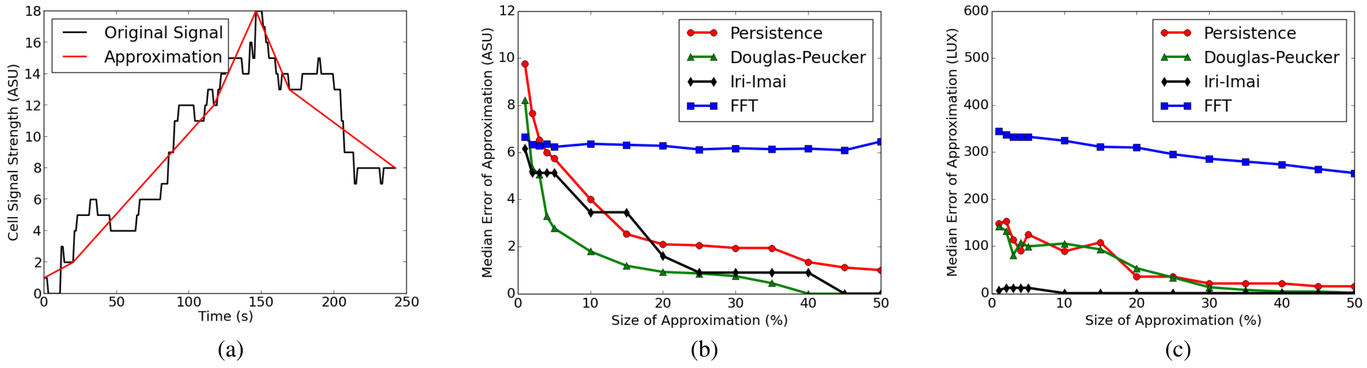

Figure 13. (a) A typical cell signal strength graph with 243 points. The red line represents the approximation at scale $s = 16$ ASU, consisting of 6 points and returning a maximum error of approximation of 3 ASU. (b) Median performance of four algorithms tested on cell signal strength data sets. (c) Median performance of four algorithms tested on light intensity data sets.

124–135, New York, NY, USA, 2004. ACM.

[10] M. Chen, M. Xu, and P. Franti. A fast o(n) multiresolution polygonal approximation algorithm for gps trajectory simplification. *Image Processing, IEEE Transactions on*, 21(5):2770–2785, May 2012.

[11] A. Collins, A. Zomorodian, G. Carlsson, and L. J. Guibas. A barcode shape descriptor for curve point cloud data. *Computers & Graphics*, 28(6):881–894, 2004.

[12] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 11(2):112–122, 1973.

[13] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete and Computational Geometry*, 28(4):511–533, 2002.

[14] M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, 2008.

[15] H.Imai and M. Iri. Polygonal approximations of a curve-formulations and algorithms. *Computational morphology*, 1988.

[16] N. Hönle, M. Grossmann, S. Reimann, and B. Mitschang. Usability analysis of compression algorithms for position data streams. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 240–249. ACM, 2010.

[17] S. Karagiorgou and D. Pfoser. On vehicle tracking data-based road network generation. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 89–98. ACM, 2012.

[18] G. Kellaris, N. Pelekis, and Y. Theodoridis. Trajectory compression under network constraints. In *Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases*, SSTD '09, pages 392–398, Berlin, Heidelberg, 2009. Springer-Verlag.

[19] M. B. Kjærgaard, S. Bhattacharya, H. Blunck, and P. Nurmi. Energy-efficient trajectory tracking for mobile devices. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 307–320. ACM, 2011.

[20] A. Kolesnikov, P. Fränti, and X. Wu. Multiresolution polygonal approximation of digital curves. In *ICPR (2)*, pages 855–858, 2004.

[21] R. Lange, F. Dürr, and K. Rothermel. Efficient real-time trajectory tracking. *The VLDB Journal*, 20(5):671–694, Oct. 2011.

[22] C. Long, R. C.-W. Wong, and H. V. Jagadish. Direction-preserving trajectory simplification. *Proc. VLDB Endow.*, 6(10):949–960, Aug. 2013.

[23] P.-F. Marteau and G. Ménier. Speeding up simplification of polygonal curves using nested approximations. *Pattern Analysis and Applications*, 12(4):367–375, 2009.

[24] N. Meratnia and R. A. de By. Spatiotemporal compression techniques for moving point objects. In E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. BÃűhm, and E. Ferrari, editors, *Advances in Database Technology - EDBT 2004*, volume 2992 of *Lecture Notes in Computer Science*, pages 765–782. Springer Berlin Heidelberg, 2004.

[25] J. Muckell, J.-H. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. S. Ravi. Squish: An online approach for gps trajectory compression. In *Proceedings of the 2Nd International Conference on Computing for Geospatial Research & Applications*, COM.Geo '11, pages 13:1–13:8, New York, NY, USA, 2011. ACM.

[26] J. Paek, K.-H. Kim, J. P. Singh, and R. Govindan. Energy-efficient positioning for smartphones using cell-id sequence matching. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 293–306. ACM, 2011.

[27] M. Potamias, K. Patroumpas, and T. Sellis. Sampling trajectory streams with spatiotemporal criteria. In *Scientific and Statistical Database Management, 2006. 18th International Conference on*, pages 275–284. IEEE, 2006.

[28] V. Radu, P. Katsikouli, R. Sarkar, and M. K. Marina. A semi-supervised learning approach for robust indoor-outdoor detection with smartphones. In *(to appear) Proceeding of The 12th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2014.

[29] S. Sankararaman, P. K. Agarwal, T. Mølhave, J. Pan, and A. P. Boedihardjo. Model-driven matching and segmentation of trajectories. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 234–243. ACM, 2013.

[30] L. Stenneth, O. Wolfson, P. S. Yu, and B. Xu. Transportation mode detection using mobile phones and gis information. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 54–63. ACM, 2011.

[31] G. Trajcevski, H. Cao, P. Scheuermanny, O. Wolfsonz, and D. Vaccaro. On-line data reduction and the quality of history in moving objects databases. In *Proceedings of the 5th ACM International Workshop on Data Engineering for Wireless and Mobile Access*, MobiDE '06, pages 19–26, New York, NY, USA, 2006. ACM.

[32] Y. Wang, H. Wei, and G. Forman. Mining large-scale gps streams for connectivity refinement of road maps. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 438–441. ACM, 2013.

[33] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining interesting locations and travel sequences from gps trajectories. In *Proceedings of the 18th international conference on World wide web*, pages 791–800. ACM, 2009.

[34] P. Zhou, Y. Zheng, Z. Li, M. Li, and G. Shen. Iodetector: A generic service for indoor outdoor detection. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 113–126. ACM, 2012.

[35] X. Zhou, S. Shekhar, P. Mohan, S. Liess, and P. K. Snyder. Discovering interesting sub-paths in spatiotemporal datasets: A summary of results. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 44–53. ACM, 2011.