Trajectory Approximation for Resource Constrained Mobile Sensor Networks

Ghulam Murtaza*, Salil S. Kanhere*, Aleksandar Ignjatovic†, Raja Jurdak†, Sanjay Jha*

*School of Computer Science and Engineering,
The University of New South Wales, Sydney NSW 2052, Australia
{gmurtaza,salilk,ignjat,sanjay}@cse.unsw.edu.au
†Autonomous Systems Lab,
CSIRO, Brisbane QLD Australia
{raja.jurdak}@csiro.edu

Abstract—Low-power compact sensor nodes are being increasingly used to collect trajectory data from moving objects such as wildlife. The size of this data can easily overwhelm the data storage available on these nodes. Moreover, the transmission of this extensive data over the wireless channel may prove to be difficult. The memory and energy constraints of these platforms underscores the need for lightweight online trajectory compression albeit without seriously affecting the accuracy of the mobility data. In this paper, we present a novel online Polygon Based Approximation (PBA) algorithm that uses regular polygons, the size of which is determined by the allowed spatial error, as the smallest spatial unit for approximating the raw GPS samples. PBA only stores the first GPS sample as a reference. Each subsequent point is approximated to the centre of the polygon containing the point. Furthermore, a coding scheme is proposed that encodes the relative position (distance and direction) of each polygon with respect to the preceding polygon in the trajectory. The resulting trajectory is thus a series of bit codes, that have pair-wise dependencies at the reference point. It is thus possible to easily reconstruct an approximation of the original trajectory by decoding the chain of codes starting with the first reference point. Encoding a single GPS sample is an \(O(1)\) operation, with an overall complexity of \(O(n)\). Moreover, PBA only requires the storage of two raw GPS samples in memory at any given time. The low complexity and small memory footprint of PBA make it particularly attractive for low-power sensor nodes. PBA is evaluated using GPS traces that capture the actual mobility of flying foxes in the wild. Our results demonstrate that PBA can achieve up to nine-fold memory savings as compared to Douglas-Peucker line simplification heuristic. While we present PBA in the context of low-power devices, it can be equally useful for other GPS-enabled devices such smartphones and car navigation units.

I. INTRODUCTION

With recent advances in embedded systems technologies, it is now possible to engineer highly compact sensor platforms. These small sensor nodes are particularly well-suited for tracking the individual movement patterns of wildlife [1]–[3]. The trajectory data stored in the node memory is offloaded to a base station (BS) when the tracked animal arrives back at a known location (e.g., watering hole). However, it is fairly typical that the animal may not return to these known locations for several days and even months (e.g., a migratory flying animal). In such instances, the sheer volume of trajectory data is likely to overwhelm the available data storage. For instance, if a GPS sample (stored as 12 bytes) is recorded every 5 minutes then a day’s worth of data requires about 3.3MB storage. If stored in its raw form, a month’s worth of data would require 100MB of storage, which is well over the capacity of typical small-form factor sensor nodes. Trajectory approximation becomes an obvious necessity for coping with this constraint. Such approximation is bound to introduce some errors in the the mobility data. However, if the error is bounded to an acceptable threshold, then the impact on the monitoring effort would be minimal. Furthermore, the compact nature of these nodes typically implies limited on-board RAM (256 - 4096 bytes) and energy constraints. This necessitates that the approximation scheme employed has low computational complexity and a small memory footprint. These constraints strongly motivate the use of an online strategy that does not require any batch processing (i.e. operates on a window of data points) and does not rely on any \textit{a priori} knowledge of the terrain.

In this paper, we consider the aforementioned problems in context of a practical application for monitoring flying foxes (also called fruit bats). Recent work [2] aims to collect fine-grained spatiotemporal data about their movement patterns and environmental surroundings by attaching a low-power sensor collar to these animals. The embedded sensor nodes record the flight and biological data such as GPS, temperature and air pressure while the bats are out and about. The data is offloaded to base stations (BS) mounted in known roosting camps when the bats return back. Figure 1 depicts a typical roosting camp and the sensor collar attached to the animal. Considering the migratory nature of these animals [4], it may be possible that data can only be offloaded after several months. The trajectory data stored on the sensor nodes can thus easily overwhelm the limited on-board storage. Thus, this application provides strong motivation and constraints for the aforementioned problem.

Online trajectory compression is an active area of research [5]–[10]. However, the particularly stringent constraints of the problem setting described above, make existing algorithms unsuitable in our context. For example, the compression algorithm proposed in [6] relies on redundancy in the underlying data to eliminate certain GPS samples. However, their scheme stores the trajectory points as raw GPS samples and thus does not exploit encoding to achieve further memory savings. Pathlet [8] requires \textit{a priori} knowledge of the spatial area covered by the mobile node.

In this paper, we present a novel online Polygon Based Approximation (PBA) technique, which approximates 2D space...
the large-scale mobile sensing scenarios with delay tolerance. The bat monitoring application as a case study, the proposed optimiz

address the fairness criteria and maximize overall network goodpu

with the constraint of network delay is presented in [17]. It establish

each super frame, which is di

in multi-hop networks focuses on fair allocation of time slots among th

proposed to achieve optimal delay, capacity gain or network utility.

tracks the direction of travel of mobile phone at the BS. They deve

In this section, we review the literature on link scheduling and optimiza

The rest of paper is organized as follows: Section 2 presents relate

section V. Finally, Section VI concludes the paper.

As explained in Section III, PBA only stores the raw value of the first GPS sample $p_1$. The remainder of the trajectory is entirely represented using codes, which are computed relative to this first sample.

We assume that the mobile node comes in contact with the base station after $n$ GPS samples have been recorded by the sensor. The encoded trajectory $T_{n}$ is then offloaded to the BS, which executes the reconstruction module (discussed in Section III) to generate an approximated GPS trajectory, $T_{n}'$, which consists of $n' <= n$ GPS samples, represented as \{p_1, p_2, p_3, ..., p_{n'}\} where $p_i$ consists of $< x', y', t' >$ tuple such that $\forall (p_i \in T_n) \exists (p_i' \in T_{n}')$ where $|\sqrt{(x' - x)^2 + (y' - y)^2}| < \epsilon_b$ and $|t - t'| < \epsilon_b$.

### III. Polygon Based Approximation (PBA)

In this section we present the details of our proposed PBA trajectory approximation algorithm. PBA only stores the first GPS sample $p_1$ in its raw form (i.e. $<\text{lat, long, time}>$). The rest of the trajectory is entirely represented using codes, which are computed relative to the first reference point (explained in detail later). PBA operates in an online fashion. For each raw sample $p_i$ recorded by the GPS sensor, PBA processes the spatial and temporal components independently. Depending on certain conditions (elaborated later in the section), there are four possibilities: (i) $p_i$ is completely eliminated from $T_{n}$, (ii) the spatial component of $p_i$ may be encoded in $T_{n}$ but the temporal component may be discarded (iii) only the temporal component is encoded while spatial component is discarded (iv) both spatial and temporal components are represented as codes in $T_{n}$.

We explain the spatial and temporal approximations separately in the first two sub-sections. This is followed by a discussion of the reconstruction scheme that is employed at the BS to reconstruct $T_{n}'$ from $T_{n}$. PBA uses a virtual grid of polygons to approximate space. Without loss of generality we use regular hexagons as a specific example in the rest of the paper. In Section III-C, we explore the implications of certain geometric artefacts on the operation of PBA, which arise from the fact that the circumsphere around a regular hexagon overlaps with its neighbouring hexagons in the grid. Finally, Section III-E discusses the implications of using polygon shapes other than hexagons on PBA.

**A. Spatial Approximation**

We represent the spatial component of PBA as PBA-SA. PBA-SA approximates 2D space using a virtual grid of regular hexagons. The reference point $p_1$ becomes the centre of the first hexagon. The rest of the grid is then constructed relative to this first sample.
to this hexagon as shown in Figure 2. The size of each hexagon is such that the distance from the centre to all the six vertices is the permitted spatial error, \( \epsilon_1^i \) meters. Thus, in PBA-SA, the hexagon becomes the smallest unit of spatial granularity. Each GPS point in the trajectory, is approximated to the centre of the hexagon containing that point. We propose a novel coding strategy for representing the hexagon in the approximated trajectory, wherein the codes represent the relative position (direction and distance) of a hexagon with respect to the preceding hexagon. This establishes a chain among the codes which is rooted at the first hexagon, for which the raw GPS coordinates of it’s centre \( p_1 \) are stored. Thus by traversing this chain in the forward direction, starting at \( p_1 \), it is possible to reconstruct an approximation of the original trajectory (explained in Section III-D).

Let \( H_i \) denote the hexagon that contains a GPS point \( p_i \). Let \( p_{i+1} \) denote the centre of this hexagon. Let \( p_{i+1} \) denote the next recorded GPS point. PBA-SA uses different encoding for following three different cases i) \( p_{i+1} \) is located within \( H_i \) (referred to as Non-Recorder) ii) \( p_{i+1} \) is located in the hexagon that is direct neighbour of \( H_i \) (Chases) iii) \( p_{i+1} \) is located in the hexagon that is two or more hops away from \( H_i \) (Jumps).

PBA-SA first computes the Euclidean distance, \( \Delta d_i \) between \( p_{i+1} \) and \( p_i \) using the Haversine formula [11]. Based on the value of \( \Delta d_i \), PBA-SA determines which of the above three conditions is true and adopts a different encoding strategy for each, as described below.

We would like to highlight that the above computation only requires the storage of 2 raw GPS samples (\( p_{i+1} \) and \( p_i \)) in the node RAM at any given time. Also, note that once a GPS point is encoded, the raw sample is discarded. Furthermore, computing the Euclidean distance is an \( O(1) \) operation. This makes our algorithm particularly well-suited for low power resource constrained nodes such as those used in the motivating application.

**Non-Recorder (\( \Delta d_i < \epsilon_1^i \))**: In this case \( p_{i+1} \) is contained within the same hexagon as \( p_i \), i.e., \( H_i \). As such, there is no need to encode this point since we have already recorded \( H_i \) within the approximated trajectory. \( p_{i+1} \) is simply discarded. Hence, the name Non-Recorder. Figure 2 provides an illustrative example. Point \( a \) is the first point, \( p_1 \) in the original trajectory. PBA-SA stores the raw sample for \( a \). The subsequent two samples, ii and iii are contained within \( H_1 \) and hence are discarded with nothing being recorded in \( T_{eb}^i \).

**Chases (\( \epsilon_1^b < \Delta d_i < 3 \times \epsilon_1^i \))**: This scenario arises when \( p_{i+1} \) is located within one of the neighbouring hexagons of \( H_i \). Figure 2 illustrates this region in shaded colour, from the point of view of \( H_i \). The neighbouring hexagons are labeled from 1 to 6 as illustrated in Figure 2 for \( H_i \). The label corresponding to \( H_{i+1} \) is used to encode \( p_{i+1} \) in \( T_{eb}^i \). Given that there are only 6 neighbours, 3 bits are sufficient to represent this code. Next we outline how we determine which neighbouring hexagon that contains \( p_{i+1} \). As shown in Figure 2, a regular hexagon can be divided into 6 equilateral triangles with the base of each triangle corresponding to one side of the hexagon. The two sides of each triangle form a fixed angle with respect to the geographic North-South line (referred to as meridian in navigation [12]). For example, the triangle containing the side labelled 2 corresponds to the range of angles 30° to 90°. PBA-SA calculates the angle between \( p_i \) and \( p_{i+1} \) with respect to the geographic North-South line using the aviation formulae by Ed Williams [13]. By comparing the result with the range of angles for the 6 triangles, one can readily determine which neighbouring hexagon contains \( p_{i+1} \). Figure 3 shows an example, where \( p_{i+1} \) forms an angle of 65° with the North-South line and is hence contained in the hexagon labelled as 2.

Since the approximated trajectory chases the original trajectory in this case, we refer to it as a Chase. The GPS coordinates for the centre of the preceding hexagon (\( p_i \)) are discarded and so is the raw sample for \( p_{i+1} \). PBA-SA now stores the GPS coordinates for the centre of hexagon \( H_{i+1} \) to determine the relative encoding of the next GPS sample.

**Jumps (\( \Delta d_i > 3 \times \epsilon_1^i \))**: This scenario arises when \( p_{i+1} \) is two or more hops away from \( H_i \). Hence named as Jump. Figure 2 shows two illustrative examples: (i) the transition from point \( c \) to the point labelled as vii and (ii) the transition from point \( e \) to point xii.

In order to record these jumps, PBA-SA makes use of the fact that it is possible to construct a right angled triangle with \( p_i \) and \( p_{i+1} \) as the two vertexes with the base along the horizontal axis of the hexagon grid. Figure 2 shows the triangle formed by points \( c \) and vii. PBA-SA encodes the jump in \( T_{eb}^i \) as the base and height of this triangle. PBA-SA uses compact codes to represent the base and height as multiples of hexagons.

---

Fig. 2. Original (solid) and approximated Trajectory (dotted) by our approximation technique.

Fig. 3. Angle representation for hexagon sides.

As shown in Figure 2, when the node move from \( a \) to \( b \), Point \( b \) is encoded as 3 and the code 011 is added to \( T_{eb}^i \) to record this chase.

**Jumps (\( \Delta d_i > 3 \times \epsilon_1^i \))**: This scenario arises when \( p_{i+1} \) is two or more hops away from \( H_i \). Hence named as Jump. Figure 2 shows two illustrative examples: (i) the transition from point \( c \) to the point labelled as vii and (ii) the transition from point \( e \) to point xii.

In order to record these jumps, PBA-SA makes use of the fact that it is possible to construct a right angled triangle with \( p_i \) and \( p_{i+1} \) as the two vertexes with the base along the horizontal axis of the hexagon grid. Figure 2 shows the triangle formed by points \( c \) and vii. PBA-SA encodes the jump in \( T_{eb}^i \) as the base and height of this triangle. PBA-SA uses compact codes to represent the base and height as multiples of hexagons.
In case, projection of \( j \) on edge-intersection, PBA-SA calculates \( j_x \) and \( j_y \) respectively. The rest of this section elaborates the details of how PBA-SA calculates and represents \( j_x \) and \( j_y \) using following equation where \( \theta \) is the angle between \( p_i \) and \( p_{i+1} \) (points \( c \) and \( vii \) respectively) as follows:

\[
\theta = \begin{cases} 
|\text{angle}(p_i', p_{i+1})| & \text{if } |\text{angle}(p_i', p_{i+1})| < 90^\circ, \\
|\text{angle}(p_i, p_{i+1})| - 90^\circ & \text{Otherwise}
\end{cases}
\]

Hence using Pythagorean theorem, PBA-SA easily calculates the length of the base and height of the triangle.

The base of the above-mentioned triangle intersects hexagons on the hexagon grid either along the edge of a hexagon or enters a hexagon from one vertex and leaves through the opposite vertex. They are referred as edge and v2v-intersection respectively as shown in Figure 4b. The occurrence of these hexagon intersection are uniquely counted towards calculation of \( j_x \) as shown in Figure 4b. Figure 4b shows that edge-intersection measures \( \epsilon_i^e \) meters while v2v-intersection is \( 2 \times \epsilon_i^e \) in length. Therefore it is important to know multiple of \( \epsilon_i^e \) that makes total base length referred as \( \alpha_{\text{hor}} = \left\lceil \frac{b_{\text{base}}}{\epsilon_i^e} \right\rceil \).

The visual representation of \( \alpha_{\text{hor}} \) is shown in Figure 4a. Whenever \( \alpha_{\text{hor}} \) is a multiple of 3, base and projection of \( p_{i+1} \) on x-axis intersect on edge-intersection, otherwise on v2v-intersection. Based on this intuition, PBA-SA calculates \( j_x \) using following equation where \( \alpha_{\text{hor}} = \alpha_{\text{hor}} - 1 \):

\[
j_x = \begin{cases} 
0 & \text{if } \alpha_{\text{hor}} < 0, \\
2 \times \left( \frac{\alpha_{\text{hor}}}{3} + 1 \right) - 1 & \text{if } \alpha_{\text{hor}} \mod(3) = 0, \\
2 \times \left( \frac{\alpha_{\text{hor}} + 3 - \alpha_{\text{hor}} \mod(3)}{3} \right) & \text{if } \alpha_{\text{hor}} \mod(3) \neq 0.
\end{cases}
\]

In order to calculate \( j_y \), PBA-SA calculates \( \alpha_{\text{ver}} = \left\lceil \frac{\text{height}}{\epsilon_i^v} \right\rceil \).

The calculation of \( j_y \) for both scenarios where base and height meet at edge or v2v-intersection differs based on the size of \( \alpha_{\text{ver}} \). Figure 4b shows both cases for triangle created by points \( c \) and \( vii \). The actual jump of 2-hops produces height of triangle to be 2 hexagons. However, it is shown that even if it was 3-hop jump instead of 2, \( \alpha_{\text{ver}} \) increases by 1 but the size of \( j_y \) stays the same. If projection of \( p_{i+1} \) falls on edge-intersection, PBA-SA calculates \( j_y \) using equation

\[
j_y = \left\lceil \frac{\alpha_{\text{ver}}}{2} \right\rceil + 1.
\]

In case, projection of \( p_{i+1} \) falls at v2v-intersection, \( j_y \) is calculated as follows:

\[
j_y = \begin{cases} 
\frac{\alpha_{\text{ver}}}{2} & \text{if } \alpha_{\text{ver}} \mod(2) = 0, \\
\frac{\alpha_{\text{ver}} + 1}{2} & \text{if } \alpha_{\text{ver}} \mod(2) \neq 0
\end{cases}
\]

Using \( j_x \) and \( j_y \), one can readily calculate \( p_{i+1} \) which is point \( d \) for jump between points \( c \) and \( viii \) in Figure 2. The reconstruction process is explained in Section III-D.

**Jump Representation in \( T_e^i \):** In case of Chases, we use 6 codes to represent neighboring hexagons from the pool of 8 possible combinations with 3 bits. PBA-SA represents \( j_x \) and \( j_y \) using the remaining 2 codes. It inserts 000 in \( T_e^i \) to represent \( j_x \) and 111 to represent \( j_y \). Whenever there is a jump for which \( j_x > 0 \) (it could be 0 as shown in Figure 2, for jump between points \( e \) and \( xii \) ) PBA-SA inserts 000 in \( T_e^i \). As shown in Figure 5, the next immediate bit in \( T_e^i \) represents the direction of \( j_x \) along x-axis (0 means it is towards right and 1 means towards left). This bit would be 0 in example jump from point \( c \) to \( viii \) in Figure 2. The next two bits in \( T_e^i \) (let us represent them by \( b, b_j \)) are used to specify the number of subsequent bits (\( \psi \)) that represent size of actual jump. As shown in Figure 5, \( \psi \) is calculated as \( \psi = 3 \times (b, b_j + 1) \). In general the jump size that can be represented by any number of \( \psi \) bits is:

\[
\text{JumpSize} = \begin{cases} 
2^\psi & \text{if } b, b_j = 0, \\
2 + 2^{\psi-3} & \text{Otherwise}
\end{cases}
\]

The counting starts from \( 2^{\psi-3} \) considering another 3 bits are only added if the previous number of bits are not enough to represent \( j_x \).

As shown in Figure 5, we use bit pattern 000111 to encode time. Therefore if direction bit for \( j_x \) is 1, \( \psi \) can not be more than 9 bits. However, if direction bit is 0, then it can be 12 bits.

In the same way if \( j_y > 0 \) then PBA-SA inserts 111 in \( T_e^i \) to represent \( j_y \). Next bit in \( T_e^i \) represents the direction of jump along y-axis (0 represent the upwards jump and 1 is for downwards). The subsequent two bits specify the number of bits that represent \( j_y \) size. PBA-SA uses the code 111000 for time encoding which is explained in detail in next subsection. Therefore \( \psi \) can be up to 9 bits for upwards \( j_y \) jump and 12 bits for downwards \( j_y \) jump.

If we encounter a jump bigger than what can be represented by \( \psi \) bit, PBA-SA inserts the maximum size jump and the remaining part of jump is inserted in \( T_e^i \) as a separate jump instance.

The example jump between points \( c \) and \( viii \) is encoded with \( j_x \) of 3 hexagons and \( j_y \) of 2 hexagons as shown in Figure 4b. The
code for this would look like \([000, 000, 010, 111,000, 001]\). First three bits (000) represent that jump has \(j_x\) component. Fourth bit (0) suggests that \(j_x\) is towards right side on x-axis. Next two bits (00) mean \(\psi = 3 \times (00 + 1)\). The value of \(\psi\) signifies that preceding 3 bits represent the actual number of hexagons to move on x-axis which is 3 (010). Next 9 bits represent a \(j_y\) of 2 hexagons in upwards direction on y-axis.

B. Temporal Approximation (PBA-TA)

As mentioned in Section III, PBA encodes the spatial and temporal components independently. This section explains the temporal encoding component of PBA, hence called PBA-TA. In order to encode time, PBA-TA makes use of allowed time error (\(\epsilon^t_b\) seconds). At any time \(t_i\), it keeps a timestamp as Current Timestamp (\(C_t\)). When \(p_i\) is recorded, PBA-TA computes the difference of \(C_t\) with its timestamp (referred as \(N_t\)) in seconds as follows, \(\Delta t = \frac{(N_t - C_t)}{1000}\). There are three possible cases based on the value of \(\Delta t\).

Case - 1 (\(\Delta t < \epsilon^t_b\)): Considering \(\Delta t\) is within error bound PBA-TA discards \(N_t\).

Case - 2 (\(\Delta t > 2 \times \epsilon^t_b\)): PBA-TA inserts time code which is 000111 (in binary) in \(T^n_{e_t}\) and updates the value of \(C_t\) to be \(C_t + 2 \times \epsilon^t_b\). All subsequently encoded GPS points are considered to be recorded at this time until PBA-TA encouters such an \(N_t\) for which \(\Delta t > \epsilon^t_b\).

Case - 3 (\(\Delta t > 2 \times \epsilon^t_b\)): Considering that \(\Delta t\) is more than double the allowed error, this case is referred as time jump. In order to to encode this case, PBA-TA inserts 111000 in \(T^n_{e_t}\). It signifies that next three bits encode the multiple calculated as \(\lfloor \frac{\Delta t}{\epsilon^t_b} \rfloor\). By definition multiple bits are only used when it is greater than 2, hence the subtraction of 3 from multiple.

C. Loose Hexagons

While using regular hexagons for approximation, \(\epsilon^b_h\) acts as radius of both hexagon and its circumcircle. An artefact of using regular hexagons as our basic approximation unit is that there could be a case where distance from centre to new GPS point \(p_{i+1}\) is less than \(\epsilon^b_h\) meters, but it is still considered as a Chase. One such case is shown in Figure 6. We refer using exact hexagons as basic unit of approximation as Strict Hexagons. Here we define notion of Loose Hexagons. When PBA-SA calculates the distance between \(p_{i-1}\) and \(p_i\). Instead of using hexagon boundaries to decide whether it is a non-Recorder, Chase or Jump, we propose to use circumference of hexagon. In this case some area between circumcircle of neighbouring hexagons will be shared as shown in Figure 6. Therefore we propose if \(p_{i+1}\) is outside the boundary of \(H_c\) but within the shared area, it should be approximated to \(p_i\) and categorised as non-Recorder instead of a Chase. An example is shown in Figure 6, \(p_{i+1}\) lies in shared area but it would still be approximated to \(p_i\). This lets us take the full advantage of base allowed error by application. We compare the performance of both variants in our evaluation.

D. Reconstruction Process

This section explains the reconstruction of approximated GPS trajectory \(T^n_{e_t}\) from coded points in \(T^n_{e_t}\). To this end, PBA reads first 12 bytes from \(T^n_{e_t}\) to get the raw values of \(<\text{lat, long, time}>\) to construct first point \((p_1)\) of \(T^n_{e_t}\). Further it traverses the chain of Chases and Jumps from \(T^n_{e_t}\) in forward direction to reconstruct \(T^n_{e_t}\).

In order to decode Chases and Jumps, \(T^n_{e_t}\) is read three bits at a time. If these three bits are between 001 and 110, it indicates a chase and specifies the label of neighbouring hexagon \(H_{i+1}\) whose centre \(p_{i+1}\) is the next point in the chain. PBA calculates \(p_{i+1}\) by using \(2 \times \epsilon^a\) and angle specified by this particular code [shown in Figure 3] as \(r\) and \(\theta\) of polar coordinates.

However, if a triplet 000 is seen that is followed by a code other than 111, it represents the start of a jump with \(j_x > 0\) as shown in Figure 5. First bit after 000 represents the direction of \(j_x\) along x-axis and next two bits \((b_y, b_z)\) indicate \(\psi = 3 \times (b_y, b_z)\). PBA reads next \(\psi\) bits to get the actual value of \(j_x\). Considering PBA-SA encodes \(j_x\) in \(\psi\) bits using equation 4, it starts the value of \(j_x\) from \(2^\psi - 3\). Therefore, in order to make use of available bits we first calculate value of \(j_x\). Let us assume that decimal value represented by \(\psi\) bits in \(T^n_{e_t}\) is denoted by \(m\). Then size of \(j_x\) that can be represented in \(\psi\) bits is calculated as:

\[
j_x = \begin{cases} m \cdot 2^\psi - 3 + m & \text{if } \psi = 3 \\ m \cdot 2^\psi & \text{if } \psi > 3, \end{cases}
\]

Further, PBA calculates length of base of the triangle using \(j_x\) and \(\epsilon^b_h\) as follows:

\[
\text{len(base)} = \begin{cases} \epsilon^b_h (j_x + \frac{j_z}{2}) & \text{if } j_x \mod(2) = 0 \\ \epsilon^b_h (j_x + (\frac{j_z}{2}) + 1) & \text{if } j_x \mod(2) \neq 0, \end{cases}
\]

PBA uses this base length and angle along x-axis (90° or -90° depending on the direction on x-axis) [see Figure 3] as \(r\) and \(\theta\) of polar coordinates to get the GPS point for projection of \(p_{i+1}\) on base of triangle.

If next 3 bits are 111 that represents \(j_y > 0\) for this particular jump. PBA calculates the length of \(j_y\) in similar manner as \(j_x\) and uses \(j_y\) to calculate the height of triangle as follows:

\[
\text{len(height)} = \begin{cases} j_y \times 2 \times \epsilon^b_h & \text{if } j_y \mod(2) = 0 \\ j_y \times 2 \times \epsilon^b_h + \epsilon^b_h & \text{if } j_y \mod(2) \neq 0, \end{cases}
\]
After calculation of \( j_p \), GPS point calculated using \( j_x \) is used as start point to calculate actually point \( p'_1 \), as result of this particular jump. PBA uses height of triangle and the angle along y-axis (0 or 180° depending on the direction) as polar coordinates to get the value of \( p_{i+1} \).

All reconstructed points are considered to be recorded at time represented by timestamp in \( p_1 \) (referred as \( C_1 \)) until we encounter a time code (000111 or 111000). When PBA encounters a time code, it adds \( 2 \times c^2 \) in \( C_1 \) in case it is 000111. If time code is 111000, PBA reads the next three bits (multiple bits) and calculates the new timestamp as \((\text{multiple} + 3) \times c^1 + C_1 \). PBA considers subsequent points to be recorded at this time until another time code is found.

E. Implications of Polygon size

Although we use hexagons to explain our approximations scheme, PBA can work with any type of regular polygons. If PBA uses a regular polygon with more side, polygon can be divided into greater number of equilateral triangles. Thus reduces the size of angle made by any two sides of the triangle. As shown in Figure 3, this angle is 60° in case of hexagon and in case we use Octagons, the angle reduces to 45°. Hence, it would introduce less error in terms of approximate direction of a Chase. However, the trade-off is that higher the number of sides of polygon, more bits are required to encode a Chase.

IV. SIMULATIONS

In this section, we outline the simulation setup and present results demonstrating the efficacy of our method.

A. Simulation Setup

In order to evaluate PBA, we have designed a custom Java simulator which accepts pre-recorded GPS trajectories as input. We use real-world traces from the flying fox monitoring project, which serves as the motivating application for our work (as discussed in Section I). The dataset consists of GPS traces from 5 distinct animals, which were gathered at different stages of the project. Table 1 summarises the properties of the dataset. The first three trajectories are from animals tagged at a common roosting camp in March 2013. The last two traces are from animals tagged at a different roosting camp in Sept. 2013. Trace 3 is the only trajectory in which GPS samples were recorded at a fixed frequency of 1 Hz. For all other trajectories, the GPS sampling frequency was varied based on the time of day. The inter-sample duration was varied from 1 second to 10000 seconds. Samples were recorded more frequently at night than during the day since the flying foxes are more active at night. We compare PBA with the Douglas-­-Peucker (DP) algorithm, which is one of the most widely used line approximation scheme. The algorithm begins with the original trajectory \( T_n \), and draws a straight line between the first and last point, represented as \( p_f \) and \( p_l \), respectively. Next it finds the furthest point within \( T_n \) from this straight line. Let \( p_m \) denote this point. If \( p_m \) is closer than \( \epsilon^2 \), then DP discards it along with all the points between \( p_f \) and \( p_l \) from the trajectory. Otherwise, the trajectory is divided in two parts: (i) \( p_f \) to \( p_m \) and (ii) \( p_m \) to \( p_l \). Then, the same procedure is applied on these two trajectories in a recursive manner. The resulting approximated trajectory is a subset of the original trajectory.

We evaluate both Loose and Strict Hexagon versions of our approximation scheme.

B. Evaluation Metrics

We use the following metrics in our evaluations.

1) Size of Approximated Trajectory: Herein, we simply compare the size of the approximated trajectory (in KB) generated by PBA with that produced by DP.

2) Hausdorff Distance: Hausdorff Distance (HD) provides the qualitative comparison of our approximation schemes with DP. Researchers have already used it for qualitative analysis of line approximation schemes [14]. Let us say that \( T \) and \( T_{\text{b}} \) are the trajectories and \( d_M(p_i, T_{\text{b}}) = \min_{p_j \in T_{\text{b}}} \|p_i - p_j\| \) where \( p_i \) is any point in \( T \). Then HD between trajectory \( T \) and its approximation \( T_{\text{b}} \) is defined as:

\[
\hat{D}_M(T, T_{\text{b}}) = \max \left( \max_{p \in T} [d_M(p, T_{\text{b}})], \max_{p \in T_{\text{b}}} [d_M(T, p)] \right)
\]

C. Results

Considering the fact that 1st and 2nd dataset are very similar in nature, we include the result of 2nd dataset only. Figure 7 shows the size of approximated trajectory obtained size of \( \epsilon^2 \) varied between 50-500 meters. Our goal of finding the roosting camps and foraging locations of flying foxes can tolerate this much error in location. One can see that our technique considerably reduces the storage requirements over original trajectory. As an example, it takes up to 103 times less space than original trajectory with \( \epsilon^2 \) of 100 meters in case of dataset-4. In all the cases shown in Figure 7, our approximation scheme achieves than 90% space reduction.

There are two fundamental reasons that underscore the improved performance of our scheme. First, since we use polygons to approximate space, all successive samples recorded within the same polygon are approximated by a single encoded entry in the approximated trajectory. Second, the codes used to represent the relative positions of the polygons achieve significant space savings, 3 bits in the best case (97% savings when compares to original 12 bytes) and 6 bytes in the worst case (50% savings when compares to original 12 bytes).

Figure 8 shows that HD of PBA is comparable with DP. Figure 8c shows a big jump in HD of trajectory produced by DP when compared to original trajectory at \( \epsilon^2 \) of 350 meters. The likely reason is that at this error value DP finds quite a large number of points in original trajectory, such that the distance between these points and \( p_m \) is less than 350
Offline methods require equivalent of O(N) RAM, whereas the most referenced Line Simplification heuristic \[15\] has been proposed in many disciplines like Computer Vision. Furthermore, offline methods include the most referenced Line Simplification heuristic called Douglas-Peucker. Offline techniques have been proposed in many disciplines like Computer Vision \[16\] and Graphical Information System \[17\]. However, these offline methods require equivalent of O(N) RAM, whereas mobile nodes have limited RAM. Our scheme on the other hand only needs to keep only two GPS points in memory.

Online Trajectory Compression techniques can be further divided into two categories. First techniques that achieve compression by eliminating redundant point based on some application specific error threshold like \[5\]–\[7, 18, 19\]. Trajecvski et al. proposes to keep location and speed information for first point \(p_1\). Further, they skip subsequent point until error is bounded by application provided threshold. If \(p_2\) is the first point that can not be approximated using application specific threshold, both speed and location of \(p_2\) are stored as well. The process continues until trajectory ends. OPW-TR \[18\] keeps a window of points in RAM and selects an anchor in that window. When the error of all the points from that anchor becomes larger, it keeps that particular point and goes on to build a new window for subsequent points. The worst case running time is \(O(n^2)\). All redundancy elimination techniques store remaining points in raw form of (lat, long, time) tuple. However, it would help to further reduce the size of trajectory if individual GPS point is represented in a compact way instead of (lat, long, time) tuple of 12 bytes.

Second category of online compression techniques perform encoding of points along with eliminating redundant points \[8\]–\[10, 20\]. In \[20\] Frchet distance is used to cluster popular subtrajectories in a given trajectory dataset. This information is further used online to cluster the recorded points. In \[8\], a pathlet dictionary is learned via solving an optimization problem. It needs to first convert the trajectories into paths on the underlying roadmap graph before running their optimization algorithms off-line and then categorises recorded GPS points based on this offline learning. All these techniques either need offline learning or some prior information of covered area as input which makes these techniques unsuitable for resource

V. RELATED WORK

Trajectory compression schemes can be categorised as online and offline compression techniques. Offline techniques include the most referenced Line Simplification heuristic called Douglas-Peucker heuristic \[15\]. Furthermore offline methods have been proposed in many disciplines like Computer Vision \[16\] and Graphical Information System \[17\]. However, these offline methods require equivalent of O(N) RAM, whereas
constrained sensor nodes.

On the other hand our technique not only combines these two, it does not need any a priori information as such. One can achieve further compression by applying lossless compression techniques like S-LZW [21] on compressed trajectory.

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed a novel GPS trajectory approximation strategy called PBA (Polygon Based Approximation) that is particularly well-suited for resource-constrained mobile sensor nodes. Our scheme abstracts two dimensional space using a virtual grid of regular polygons. The size of the polygons are determined by the allowed spatial approximation error and the virtual grid can be readily constructed starting with the first point in the trajectory. Each GPS sample is approximated to the centre of the polygon that it is contained within. PBA only stores the raw GPS sample for the first point in the trajectory. A novel encoding strategy is proposed that uses bit codes to represent the relative position of the polygons that contain two successive points in the trajectory. PBA creates a chain of dependencies among successive codes. It is thus possible to easily reconstruct an approximated version of the trajectory which is bounded by the allowed spatial error. PBA achieves an $O(n)$ complexity and incurs a very small memory footprint, thus making it well-suited to low power sensor codes. However, our scheme can be readily used for online trajectory compression on other GPS-equipped mobile devices such as smartphones and car navigational units. PBA is evaluated using real mobility data from sensor nodes attached to flying foxes in the wild. Our results demonstrate that PBA can achieve up to nine-fold memory savings as compared to one of the most widely used trajectory compression scheme.

In our future work, we plan to implement PBA on low power sensor nodes and conduct field trials in the context of the motivating application described in this paper. We also plan to research an extension to PBA that will permit a node to recompress an already approximated trajectory on the fly when faced with the possibility of exhausting the available storage. Further, we also intend to explore the possibility of merging trajectories from multiple nodes in an ad-hoc manner.

VII. ACKNOWLEDGMENTS

This work was supported by the Australian Research Council Discovery Grant DP110104344. We thank Dr. Goce Trajcevski and Dr. Branislav Kusy for constructive discussions in relation to this work.

REFERENCES


