Hierarchical Temporal Mobility Analysis

Adele Hedrick
Faculty of Science
UOIT
adele.hedrick@uoit.ca

Ken Q Pu
Faculty of Science
UOIT
ken.pu@uoit.ca

Ying Zhu
Faculty of Business and IT
UOIT
ying.zhu@uoit.ca

ABSTRACT

It's becoming a commonly accepted standard of living that we constantly have access to a mobile device which is basking in a sea of Wifi hotspots. By recording the observed Wifi signals over time, it is possible for the mobile phone to deduce the salient locations in its environment, and the mobility patterns of the user. In real-life, fluctuations of the Wifi hotspots and the unreliability of mobile phone antenna necessarily creates false readings and missed readings, making the location identification problem and its related problems particularly challenging.

In this paper, we propose a family of algorithms to perform the tasks of location identification, mobility inference, and localization. Our algorithms are able to handle the noisy reading observed in real-life application. Furthermore, our location identification algorithm constructs a hierarchy of salient locations providing a multiresolution model of the environment.

Keywords

ACM proceedings; Mobility; Temporal Analysis; Personal Data

1. INTRODUCTION

With the omnipresence of WiFi hotspots throughout the urban and suburban environments, it has become possible to perform location identification and mobility analysis on any mobile devices. Given the nature of WiFi networks, sensing the surrounding WiFi basic service set identifiers (BSSID) can be done accurately with very little power requirement and latency, making it an ideal way generate spatio-temporal mobility data streams. The *raw* mobility data stream consists of a time series of *readings*. Each reading has a timestamp, and a collection of triples $\langle \text{bssid}, \text{ssid}, \text{strength} \rangle$, where bssid is a univerally unique identifier of a WiFi hotspot, ssid is a user-defined name, and finally the strength is a numerical measure of the intensity of the signal of that particular WiFi hotspot.

WebDB'16 San Francisco, USA
© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

The problem of location inference based on WiFi scans has received a great deal of attention (e.g. [**?**]). Our work distinguishes from the existing literature in several important ways. We do make any assumptions on the signal propagation model, nor any prior knowledge on the ambient environment. In the *location identification phase*, our algorithm continuously constructs a set of physical locations based on the observation of the BSSID's and their signal strength. Our emphasis is that location identification is performed in an unsupervised, ad-hoc, and incremental fashion. To make it suitable for the mobile runtime environment, the *online* algorithm runs with sublinear time and space-complexity.

Our algorithm consists of a series of pipelined stages of WiFi reading processing. Each stage is a non-blocking online algorithm whose output is consumed by the next stage.

Movements

In the first stage, we detect *movements* of the mobile device by maintaining a multiresolution timeline of the readings. We hierarchically partition the timeline to detect mobility at various time scales. The multiresolution is used to reject false positives and cope with false negatives in the WiFi readings.

Physical Locations

The segments generated by the movement detection phase are further processed to build a database of *physical locations* based on the BSSID signatures of the segments. This allows us to map out the ambient environment over time. We note that movement detection and physical location identification are both non-blocking, so that physical locations are identified instantly, and incrementally refined.

Semantic Clustering

While the BSSID information provides accurate physical location identification, they do not carry sufficient *semantic* value to the end user. For the sake of minimizing power consumption, we do not wish to utilize the GPS capability of the mobile device. It turns out that the user-defined SSID names of the WiFi hotspot are a useful source of semantic knowledge of the physical locations. Our algorithm analyzes the SSID names to organize the physical locations into a semantically meaningful hierarchy.

Location Inference

Finally, the algorithm uses maximal likelihood estimation to infer locations with semantic labels to the raw WiFi readings.

2. RELATED WORK

Radio frequency and ultrasonic signals are used in conjunction [**?**] to estimate the location of mobile devices, with the overhead of installing beacon devices throughout the building and listener devices to user mobiles. This method only works in indoor environments and requires pre-deployment equipment installation. Relying only on WiFi signals though does not require any pre-deployment overhead since WiFi access points already exist everywhere. Received signal strength indication (RSSI) measurements between the cellular base stations and the user device [?] are processed by algorithms combining an extended Kalman filter, approximate pattern matching and velocity vectors to predict future movements of users across cell boundaries. It is suitable for only outdoor environments and does not apply naturally to small-scale user mobility, e.g., indoor inter-room movements. In works such as [?, ?], the indoor area is divided into a grid and signal data is recorded from a set of fixed known points which is used as training data, generating a probability distribution of signal strengths given location values. With new signal observations, the posterior distribution is computed and the location with the highest probability is chosen.

The works of [?] and [?] both model user locations as states of a dynamic system with the noisy observations of RSSI data, and apply an implementation of Bayesian filters called particle filters to probabilistically estimate the system state (i.e., user location). An initial training phase builds a wireless sensor map of the environment divided into cells by sampling at predefined points; this step is dependent on the assumed WiFi sensor model, either signal propagation or fingerprinting models. Then location is estimated on a spatial connectivity graph; each motion update step of the Bayesian filter moves the user along an edge of this graph. These probabilistic methods are powerful but suffers the drawback of potential high computational complexity inherent in the particle filters method whose worst-case complexity grows exponentially in dimensions of the state space. Moreover, all these above-mentioned approaches all require an initial training phase, and therefore are not online and ad-hoc.

3. MOVEMENT SEGMENTATION

We study the problem of online processing of WiFi scans collected by a mobile device. We refer to each scan as a *reading*. A reading is defined as $\langle t(r), \mathbf{B}(r), \mathrm{BSSID}, s(\cdot|r)\rangle$ where $t(r)$ is the timestamp of the reading, and $\mathbf{B}(r) \subseteq$ BSSID is a set of BSSID of the wifi hotspots that the scan detected. The BSSID : $\mathbf{B}(r) \to$ Names is a mapping of BSSID names to user-defined name of the WiFi hotspot. $s(\cdot|r) : \mathbf{B}(r) \to R^+$ is a mapping of BSSID of the reading r to a signal strength; $s(b|r)$ is the intensity of b in the reading r.

3.1 Movement detection

In this section we describe an online algorithm to organize the timeline of readings into multiresolution segments. The timeline is partitioned such that within each segment,

Definition 1. *A* timeline T *is a sequence of readings. We denote* T_i *as the i-th reading of the timeline* T.

A segment *of the timeline* S *is a contiguous subsequence of* T.

The $\mathbf{B}(T_i)$ induces a similarity measure among the readings.

```
func TimelineClustering($T$)
$X = T$
while $|X| > 1$ {
    $i^* = \mathrm{argmax}\{\mathrm{sim}(T_i, T_{i+1}) : i \in [1, |X|]\}$
    $r = \mathbf{merge}(T_i, T_j)$
    $X = \mathbf{replace}\ [T_i, T_{i+1}]\ \mathbf{with}\ r$
}
return Tree with root $X$
```

Algorithm 1: Bottom-up timeline clustering

Definition 2 (Reading similarity).

$$\mathrm{sim}(T_i, T_j) = \mathrm{Jaccard}(\mathbf{B}(T_i), \mathbf{B}(T_j)) = \frac{|\mathbf{B}(T_i) \cap \mathbf{B}(T_j)|}{|\mathbf{B}(T_i) \cup \mathbf{B}(T_j)|}$$

This allows us to organize T using hierarchical clustering. Unlike the traditional hierarchical clustering of sets of items with a similarity measure, clustering a timeline has the added constraint that each cluster must only contain adjacent readings. Algorithm 1 is an adaptation of the bottom-up agglomerative to compute hierarchical timeline clustering. In **TimelineClustering**(T), $\mathbf{merge}(T_i, T_{i+1})$ creates a new node in the hierarchical timeline, and its BSSID set is simply the union of the BSSID sets of its children:

$$\mathbf{B}(\mathbf{merge}(x, y)) = \mathbf{B}(x) \cup \mathbf{B}(y)$$

We remark that **TimelineClustering** is certainly *not* an online algorithm, as it requires $\mathcal{O}(|T|^2)$ complexity to build the hierarchy. This will be remedied later by the online version to be presented later.

Let $H = \mathbf{TimelineClustering}(T)$, where H is a binary tree with the leaf nodes as the readings in T. H presents a multiresolution segmentation.

We use the following notations:

- The leaf-nodes of H, written leaf(H), is just the readings T. The nodes of H is writtn Nodes(H).
- The descendants of $v \in$ Nodes(H) is written descendants(v).
- The readings of v is defined as:
$$\mathbf{R}(v) = \mathrm{descendants}(v) \cap \mathrm{leaf}(H)$$
- Since H is a binary tree, each interior node v has two children, written left(v) and right(v).

level of v.

The challenge is the determine the nodes in H whose readings are all at the *same* physical location. This is determined by the *minimum similarity.*

Definition 3 (Minimum Similarity). *Given a set, R, of readings, the minimum similarity of R, written* minsim(R) *is defined as:*

$$\mathrm{minsim}(R) = \min\{\mathrm{sim}(r, r') : (r, r') \in R \times R\}$$

The minsim provides a measure if a set of readings R span over more than one physical location. We observe that if R contains readings taken from two distinct physical locations, then, due to the spatial distance there will be at

```
func Segments($v, \epsilon$)
if minsim($v$) $> \epsilon$ {
    return [$v$]
} else {
    Segments(left($v$), $\epsilon$) $\cup$ Segments(left($v$), $\epsilon$)
}
```

Algorithm 2: Identifying homogeneous nodes

least one pair of readings, r and r', that are quite dissimilar: $\mathrm{sim}(r, r') \leq \epsilon$ for some small constant $\epsilon \ll 1$. This means that $\mathrm{minsim}(R) \leq \epsilon$.

For nodes in the hierarchy, $v \in \mathrm{Nodes}(H)$, its minsim is simply the minsim of its readings.

Definition 4 (Minsim for nodes).

$$\mathrm{minsim}(v) = \mathrm{minsim}(\mathbf{R}(v))$$

We will use minsim and ϵ to perform the segmentation of T using the hierarchy H. Algorithm 2 performs a top-down traversal of H to identify the top-most nodes in H with $\mathrm{minsim}(R) > \epsilon$, which we call *homogeneous nodes*.

Each homogeneous node in corresponds to a segment of the timeline in which all the readings are at the same physical location according to the BSSIDs.

There are two shortcomings with the approach:

1. The timeline clustering is not an online algorithm.

2. The segmentation requires the threshold measure ϵ.

We will address these issues in the coming sections.

3.2 Online hierarchical timeline clustering

Our objective is to update a hierarchical timeline cluster H incrementally when more readings are appended to the timeline T.

We define a procedure $\mathrm{update}(H, v_{\mathrm{insert}}, v_{\mathrm{new}})$ where H is the timeline cluster to be updated, and we wish to add a new node v_{new} *after* the v_{insert}.

We analyze the behaviour of update(...) by different cases. Let $H' = \mathrm{update}(H, v_{\mathrm{insert}}, v_{\mathrm{new}})$

- If $v_{\mathrm{insert}} = \mathrm{root}(H)$, then $H' = \mathrm{newnode}(v_{\mathrm{insert}}, v_{\mathrm{new}})$

- Otherwise, define $v_{\mathrm{parent}} = \mathrm{parent}(v)$, and $v_{\mathrm{prev}} = \mathrm{left}(v_{\mathrm{parent}})$.

  Next week check if v_{new} is closer to v_{insert} or not, compared to v_{prev}.

  - If $\mathrm{sim}(v_{\mathrm{new}}, v_{\mathrm{insert}}) \geq \mathrm{sim}(v_{\mathrm{insert}}, v_{\mathrm{prev}})$, then we create a new node $v' = \mathrm{newnode}(v_{\mathrm{insert}}, v_{\mathrm{new}})$, and promote v_{prev} to replace v_{parent}. Then, we recursively invoke:

  $$H' = \mathrm{update}(H, v_{\mathrm{prev}}, v')$$

  - If $\mathrm{sim}(v_{\mathrm{new}}, v_{\mathrm{insert}}) < \mathrm{sim}(v_{\mathrm{insert}}, v_{\mathrm{prev}})$, then we try to insert v_{new} after v_{parent}.

  $$H' = \mathrm{update}(H, v_{\mathrm{parent}}, v_{\mathrm{new}})$$

3.3 Unsupervised segmentation

In practice, the WiFi readings will encounter several diferen types of *noises*.

- False positives - it is possible that a reading will detect one or more BSSIDs which are not local to the physical location. False positives may be the result of faulty WiFi scan, or by moving WiFi hotspots created by other mobile devices.

- False negatives - a reading my miss certain BSSIDs which are local to the physical location. This may be the result of signal blockage or momentary interference.

- Moving BSSIDs - moving mobile devices often create transient WiFi hotspots, making it possible for a common WiFi BSSID appear at multiple physical locations.

By choosing the right threshold during the segmentation ϵ, we can eject errors due to noise. With the right choice of ϵ, we classify readings with sufficient difference (as low similarity between their BSSID sets) as distinct locations even if some readings contain errorenous BSSIDs.

We make the assumption that all the segments in H are either homogeneous or not. At the leaf level, we have $\mathrm{minsim}(v) = 1$, and as we traverse upwards toward the root, $\mathrm{minsim}(v) \to 0$. A sudden drop in minsim from level i to $i+1$ suggests that at the higher level (and coarser timeline partition), nodes are covering more than one physical location.

The selection of ϵ is minsim corresponding to the level with the largest drop in average $\mathrm{minsim}(v)$ for v in the level.

4. LOCATION IDENTIFICATION

In Section 3, we described a unsupervised online algorithm to identify a sequence of movements, $\mathbf{S}$, based on the BSSID readings. The result of the algorithm is a sequence of segments, each of which is guaranteed to be present a single physical location.

In this section, we describe an efficient algorithm to identify the set of *unique* locations based on BSSID signatures of each segment. Given a segment (a sequence of consecutive readings in the timeline) S, $\mathbf{R}(S)$ is the readings belonging to S, and $\mathbf{B}(S) = \bigcup\{\mathbf{B}(r) : r \in \mathbf{R}(S)\}$.

The unique locations are characterized by repeated occurrences of segments with similar BSSID signatures.

While we can use k-mean clustering to cluster the segments based on the BSSID signatures, our application requies an efficient online method that does not require a prescribed value of k.

Recall that the segments $\mathbf{S}$ is being produced in a streaming fashion. will propose an algorithm to incrementally build a database of physical locations $\mathbf{L}$. Each physical location $L \in \mathbf{L}$ is characterized by a set of *weighted* BSSIDs.

The algorithm begins with an empty set. As a segment S is generated in $\mathbf{S}$, we look for a location $L \in \mathbf{L}$ that is sufficiently similar S by the measure of $\mathrm{sim}(\mathbf{B}(L), \mathbf{B}(S))$. If no such L exists, then we have discovered a new location, and we create a new location in $\mathbf{L}$ based on $\mathbf{B}(S)$.

The algorithm of location identification is given in Algorithm 3. It has a parameter c which is the minimal similarity for a segment to belong to a physical location.

```
func LocIdentify($\mathbf{S}$)
$\mathbf{L}$ = new index of locations
foreach $S \in \mathbf{S}$ {
    find $L \in \mathbf{L}$ such that $\mathrm{sim}(\mathbf{B}(L), \mathbf{B}(S)) > c$
    if not found {
        add($\mathbf{L}$, newloc($S$))
    } else {
        update-loc($L$, $S$)
    }
}
```

Algorithm 3: Algorithm for identifying distinct locations from a stream of movements.

```
func SemanticMerge($\mathbf{L}$, n)
$K^* = \mathrm{argmax}\{\mathrm{reduction}(\mathbf{L}, K) : K \in \mathrm{keys}(\mathbf{L}, n)$
$\delta = \mathrm{reduce}(\mathbf{L}, K)$
while $\delta > 0$ {
    $\mathbf{L} = \mathrm{merge}(\mathbf{L}, K)$
    $K^* = \mathrm{argmax}\{\mathrm{reduction}(\mathbf{L}, K) : K \in \mathrm{keys}(\mathbf{L}, n)$
    $\delta = \mathrm{reduce}(\mathbf{L}, K)$
}
return $\mathbf{L}$
```

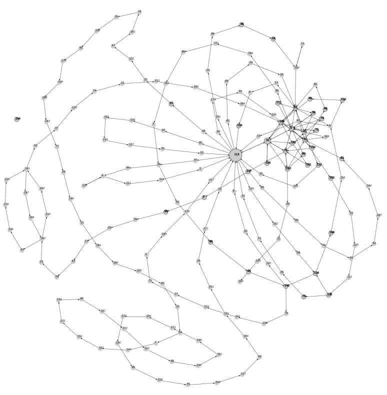
Algorithm 4: An algorithm to aggregate physical locations to groups based on their semantic information.

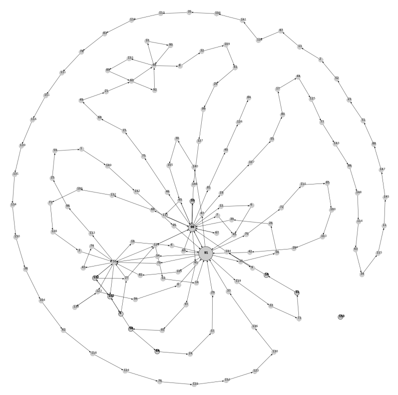
5. SEMANTIC GROUPING OF PHYSICAL LOCATIONS

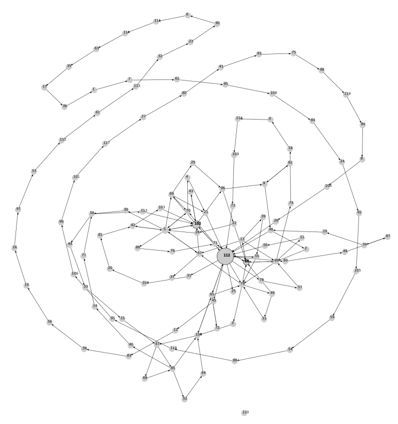
6. EXPERIMENTS

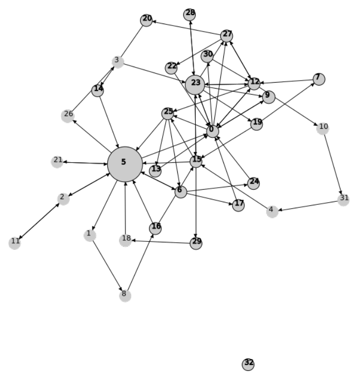
```
func merge($\mathbf{L}$, $K$)
$L_{\mathrm{new}} = \mathrm{newlocation}(\emptyset)$
foreach $L \in \mathbf{L}$ {
    if $K \subseteq \mathrm{BSSID}(L)$ {
        $L_{\mathrm{new}} = L_{\mathrm{new}} \cup L$
        delete $L$ from $\mathbf{L}$
    } }
$\mathbf{L} = \mathbf{L} \cup \{L_{\mathrm{new}}\}$
return $\mathbf{L}$
```

Algorithm 5: Merging locations based on a key SSID set.

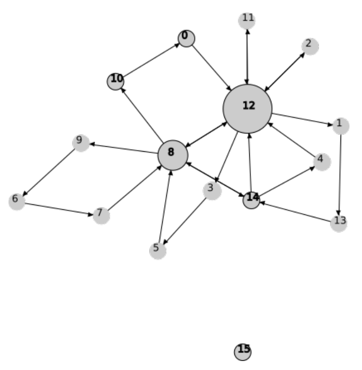

(a) L0


(b) L1

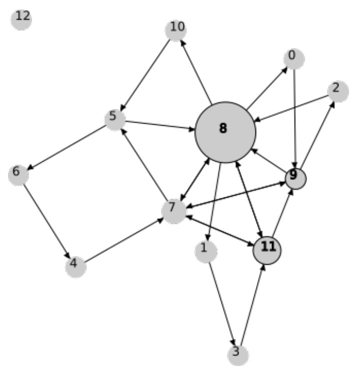

(c) L2



(d) L0 filtered



(e) L1 filtered



(f) L2 filtered