RoutineSense: A Mobile Sensing Framework for the Reconstruction of User Routines

Jean-Eudes Ranvier Michele Catasta Matteo Vasirani Karl Aberer
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland
{firstname.lastname}@epfl.ch

ABSTRACT

Modern smartphones are powerful platforms that have become part of the everyday life for most people. Thanks to their sensing and computing capabilities, smartphones can unobtrusively identify simple user states (e.g., location, performed activity, etc.), enabling a plethora of applications that provide insights on the lifestyle of the users. In this paper, we introduce routineSense: a system for the automatic reconstruction of complex daily routines from simple user states, implemented as an incremental processing framework. Such framework combines opportunistic sensing and user feedback to discover frequent and exceptional routines that can be used to segment and aggregate multiple user activities in a timeline. We use a comprehensive dataset containing rich geographic information to assess the feasibility and performance of routineSense, showing a near threefold improvement on the current state-of-the-art.

Categories and Subject Descriptors

I.5.1 [**Pattern Recognition Model**]: Model

Keywords

activity modeling, routine detection

1. INTRODUCTION

The ever-increasing sensing and computing capabilities of modern smartphones are enabling new unobtrusive ways to collect the digital trails of users, organise them and extract meaningful information regarding their activities. Thanks to sensors such as GPS, accelerometer and microphone, several user-centric smartphones applications have been recently developed to sense and understand the way we live and the world that surrounds us [1]. These applications typically focus on recognising specific aspects of the user life and creating long sequences of user-related states, such as the physical activity (e.g., walking, standing, running, etc.) or the visited locations (e.g., home, workplace, etc.). However, how to identify complex patterns, or routines, from sequences of simple user states is still an ongoing challenge. Applications that could benefit from routineSense range from elder care with the monitoring of patients' routines, to personalized recommendations (e.g., fitness advices, activity suggestions, reminders, etc.) based on the user's routine.

In this paper, we introduce an incremental processing framework for fusing sensors data into simple states, which can be further aggregated into complex routines. The proposed framework combines opportunistic sensing, unsupervised pattern generation and user feedback to discover and rank frequent and exceptional routines in an incremental way, following an established human memory model. We evaluate our framework using a comprehensive data set containing rich geographic information. The evaluation is both in terms of computational requirements of the routine reconstruction and accuracy of the segmentation of the sequence of states. Furthermore, we also implement a state-of-the-art algorithm to assess the accuracy of our approach.

The contributions of this paper are the following: (i) we define a theoretical framework for the generation of simple states and the reconstruction of complex patterns based on the cognitive model of episodic memory proposed by Conway [2], according to which single atomic events, or episodic elements (EE), are grouped into simple episodic memories (SEM); (ii) we propose two models based, respectively on frequent pattern mining and on finite state machines, to generate in an unsupervised way sets of SEMs (each of them corresponding to complex patterns) from sequences of EEs.

The organization of the rest of the paper is as follows. In Section 2, we present approaches taken for related problems. In Section 3, we present the details of our framework and more specifically in Section 4, we describe the two approaches taken to generate routines. In Section 5, we evaluate these two approaches against a real life dataset and compare their accuracy again the T-pattern algorithm. We conclude with a summary in Section 6.

2. RELATED WORK

2.1 Mobile Activity Recognition

Mobile sensing and mobile activity recognition have already been studied in multiple context. In [3], Zheng et al. describe a supervised learning approach to augment a timeline of raw GPS coordinates with the different inferred transportation modes. This system is part of a larger project [4] which proposes storage and search mechanism for these aug-

mented GPS trajectories. The transportation mode detection problem is also studied in [5]. A transfert of technology resulting from these research works enabled an implementation of the transportation mode detection in recent mobile operating systems such as the activity recognition API of Android.

Several approaches have been taken in [6], [7], [8] and [9], to structure the data gathered via mobile sensors and to infer the context in which this data was acquired, in term of activity, location, etc. In particular, [10] proposes the creation of a diary of the whereabouts of the user based on GPS traces. Closely related, in [11], the authors develop a diary containing the stay points of the users collected from smartphones and augmented with external information from Google map and Yelp in order to allow for searchable locations. In [12], Guo et al. propose to harness mobile sensing in order to improve the recall of human memory. To this purpose, they extract contact information and memory cues from physical and virtual sensors and structure these contact information in order to facilitate their recall through the appropriate cues. However, all these operation are usually performed on single or pairs of activities. Abstracting groups of activities into routines allows us to discover larger scale patterns.

2.2 Bounded-States Routine Detection

The aggregation of simple activities into high-level routines has also been explored in previous works. In [13], the authors propose a method to discover complex activities based on smart home sensors. They use a latent Dirichlet allocation(LDA) and frequent sequence mining algorithm to build complex activities. In this context, the number of sensors and the number of activities that can be tracked are limited. This limitation allows the authors to fix beforehand the number of topics used in LDA. In [14], the authors also aim at detecting a finite set of activities based on accelerometer data. The K-mean algorithm is used first to cluster sensors data into low level events which are then processed by a boosted set of classifiers to determine which high level activity is performed. However, K-mean necessite the apriori knowledge of the number of events that can be detected, making this method not suitable for problems in which the number of states or activities is unbounded. The authors of [15] and [16] use a similar representation of states (home, work, out, no information) to perform a 4-class classification. Farrahi et al. [15] segment the user's location trace into a timeseries of labels before modeling it using LDA augmented with the author-topic model, to define routines. Conversely, Eagle et al. [16] capture the behavior of users using the notion of *eigenbehaviors*, the principal components of a behavioral space. Once again, these approaches are restricted to the detection of a finite set of states. While combinations of these states can lead to multiple routines, it does not allow for a fine grain distinction between them.

2.3 Unbounded-States Routine Detection

The notion of T-pattern presented in [17] describes frequent behaviors both in term of space and time. T-patterns are mined from GPS traces by first establishing a list of dense regions traversed by the users. These regions are then analysed from a temporal perspective in order to extract frequent temporal patterns. This method accepts an unbounded number of states and can generate an infinite number of patterns. However, this method focuses on the GPS representation of the data and therefore does not benefit from additional semantic information. Furthermore, it focuses on sequential patterns, disregarding potential temporal swapping of events.

The same way the human brain structures memories in a hierarchical fashion, we propose a framework which aggregates low level sensor readings into more abstract multimodal states which can be, in turn, aggregated into high level representation of the user routine. It improves upon existing work by allowing for an unbounded set of states defined by their semantic representation. These states can then be aggregated into an unbounded set of routines. The algorithms used by the framework are essentially incremental. This allows for periodic processing of the increments as well as energy efficiency, which are both required for mobile applications.

3. ARCHITECTURE

RoutineSense generates a high-level representation of a user's life based on his smartphone sensor readings. Inspired by the episodic memory model proposed by Conway [2], we structure our framework based on his representation of the human memory responsible for storing and recollecting past experiences. Figure 1 depicts the resulting approach taken by routineSense.

3.1 Episodic memory model

The atomic elements of the episodic memory model are episodic elements (EE). They represent events or summary of events. We translate this concept into the digital world as a 3-dimensional entity containing the answers to the questions *When?, Where?* and *What?* (e.g. *At 10am* I *went jogging* by the *lake side*). The *when* relates to the temporal nature of the memories and correspond to the instant at which an event occurs. It is naturally represented in our system as the timestamp at which this event happened. The *where* corresponds to the location at which the event happened. While GPS coordinates would be a natural candidate to represent this dimension, they lack interpretability. We propose to augment these coordinates with a semantic representation of the venue. The *what* represent the activity being carried out by the user. The representation of the activity also follows a semantic approach setting the base for comparison and querying of activities.

Still according to Conway's model, episodic elements can be aggregated into simple episodic memories (SEM). These entities represent higher level activities and last a longer timespan (e.g. going to a restaurant and then going to the opera can be aggregated as going on a date). We emulate this behavior in our system by allowing the user to manually group EE to create SEM and by discovering recurring patterns of EEs in the past activities. The advantage of this approach is that EEs and SEMs encourage composability of memories, while at the same time the cognitive burden while interacting with the system is reduced, hence facilitating higher memory recall.

3.2 Generation of Episodic Elements

RoutineSense defines two types of episodic elements. The first one is related to the opportunistic sensing of the user's whereabouts. Opportunistic episodic elements contains the following information: the time of the activity, the staypoint

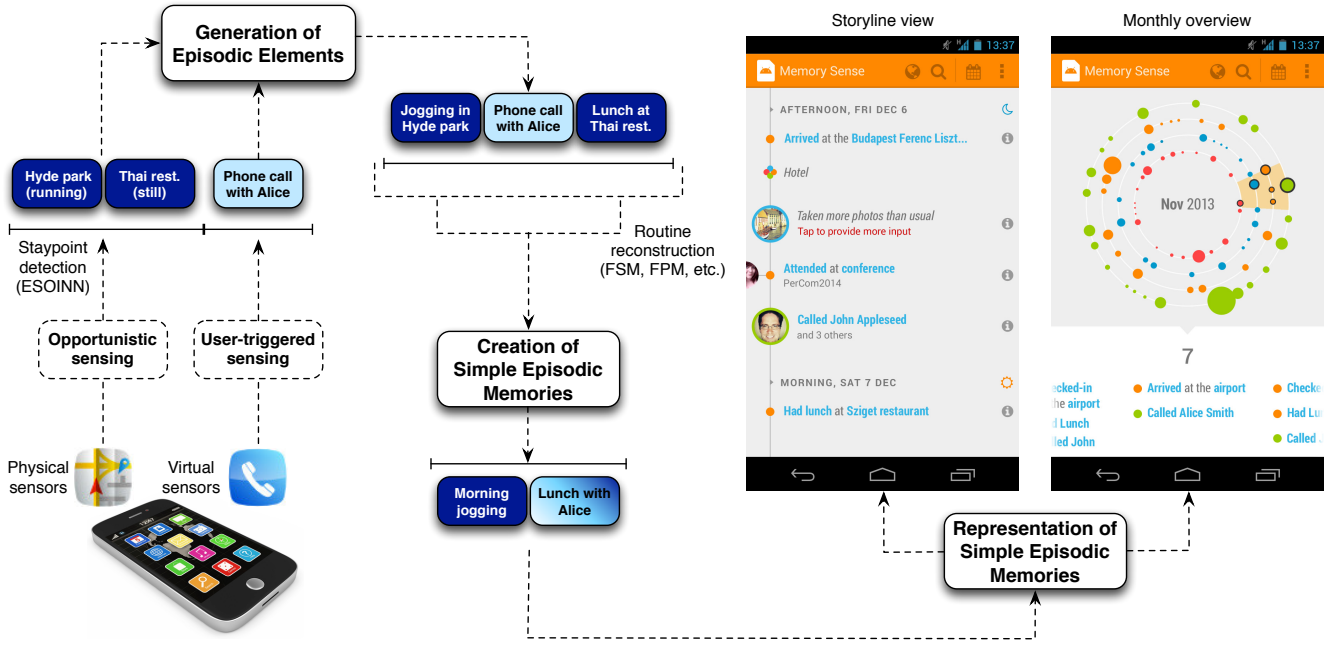

Figure 1: ***The architectural pipeline of routineSense.*** **First, the Episodic Elements are generated by an algorithm for staypoint detection (based on ESOINN), and by the virtual sensors in the phone (e.g., phone calls, SMS, etc.) Then, the routine are reconstructed directly from the Episodic Elements. Finally, our prototype application (embedding routineSense) shows the routines at different granularity levels.**

at which the event occurs and the description of the activity the user is performing.

The time of the activity is represented by the timestamps at which the activity started and ended. Staypoints are defined as the locations at which the user stayed for a minimum amount of time to perform an activity. They are generated based on a finite state machine using geofencing and GPS data to determine the state of the user. Due to the imprecision in GPS readings and the continuous nature of the spatial domain, staypoints need to be clustered in order to detect recurrent visits of the same place. The clustering will allow in a second time to avoid having to characterize a new staypoint if it belongs to an already characterized cluster. This problem is handled by using the ESOINN algorithm [18] to cluster staypoints. ESOINN possesses several suitable features to this purpose such as being incremental, unsupervised and does not require to know apriori the number of clusters. The output of the algorithm represents the unique places visited by the user. However, some modifications were required to better fit the needs of routineSense. These modifications include the removal of the forgetting mechanism of ESOINN in order to retains all the data, modification of the density function in order to simplify the computation, therefore making it suitable for mobile computing, and generation of consistent cluster labels to reuse from one increment to another. Once the staypoints are acquired and clustered, we enrich them semantically by attributing a venue to each cluster. The attribution is done by querying location-based services such as Foursquare or Google places to get a list of the k-closest venues. The venues are then ordered based on the frequency of past visits of the user. While this order requires a bootstrap phase involving user feedback, it will eventually converge to a user-specific ordering. The description of the activity is composed of a verb and a complement. In order to semantically structure the description, verbs and complements are selected from the wordnet database. This approach allows for consistency in the descriptions and facilitates the definition of a distance between two description. As for the staypoints characterization, routineSense makes use of the history of the user's activities to generate a list of potential descriptions based on the frequency of activities at the specific staypoint.

The second type of episodic elements generated by routineSense is based on the virtual sensors of the phone. They represent the interaction that the user had with the phone and are therefore called user-triggered. They encompass calls and sms received and given by the user, calendar events, pictures taken and social interactions. Some of these EEs are not necessarily geo-localizable (i.e. sms, calls) and therefore it would be difficult to attribute them a meaningful location. However, they all contain a timestamp and an activity, which allow us to fuse them in the timeline therefore diversifying the information collected about the user.

4. GENERATION OF SIMPLE EPISODIC MEMORIES

Once the EEs are generated, they can be aggregated, according to the episodic memory model, into simple episodic memories. When a new SEM is created, the user can provide the framework with a label which will be reused when posterior instances of the SEM are detected. In the framework presented in this paper, we fix the granularity of SEMs by assuming that they have the timespan of a day. This assumption simplifies the research problem at hand while keeping it realistic: weddings, week days, excursions can be seen as

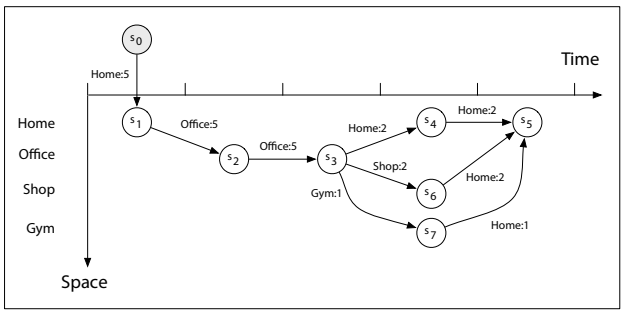

Figure 2: Example of finite state machine.

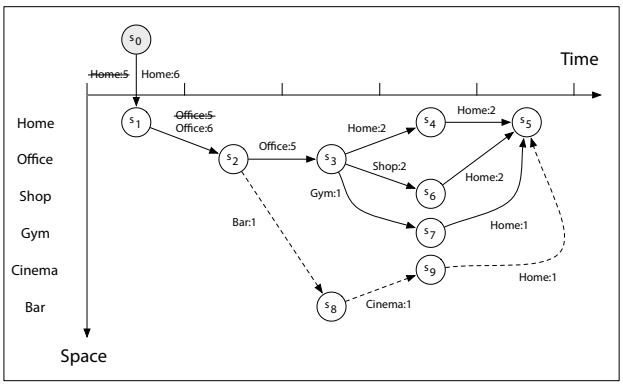

Figure 3: Example of matching of a sequence of EEs against an existing finite state machine.

an aggregation of multiple EEs over a day. The generalization of SEM of variable length is left as future work. In the following section we propose two concurrent approaches to generate SEMs. The first one is based on Finite State Machine generation while the second one leverages on frequent itemset mining and frequent sequence mining.

4.1 Finite state machine model

Finite state machine (FSM) is a mathematical model designed to model different states and the possible transition between these states. The first step towards the creation of simple episodic memories using the FSM model is creating sequences of episodic elements, corresponding to one day of the life of the user. To do that, we discretize the time dimension into k buckets. For each bucket, there is a single episodic element, corresponding to the most representative activity occurred in the time bucket. In case that more than one episodic element have been generated for that bucket, we select the most representative one with the majority rule. Therefore, at the end of one day in the user life, a sequence $\{ee_1, ee_2, \ldots, ee_k\}$ is generated by the system. Each episodic element ee_j is defined as $< \ell, t >$, where $\ell \in \mathcal{L}$ is the location type, $\mathcal{L}$ is the set of location types, and $t \in \mathcal{T} = \{1, \ldots, k\}$ is the time bucket.

A simple episodic memory (SEM) is modeled as a finite state machine $< \mathcal{S}, s_0, \Sigma, \Lambda, T >$, where $\mathcal{S}$ is the set of states, s_0 is the initial state, Σ is the input alphabet, Λ is the output alphabet, and $T : \mathcal{S} \times \Sigma \rightarrow \mathcal{S} \times \Lambda$ is the transition/output function. In our modelling, $\mathcal{S}$ is the set of all possible location type-time pairs, $\mathcal{S} \subseteq \mathcal{L} \times \mathcal{T}$, the input alphabet Σ is the set of location type $\mathcal{L}$, and the output alphabet Λ is the set of integer values $\mathbb{Z}$, representing the number of times a specific location type have been observed when transiting from one state to another. Figure 2 shows an example of FSM, corresponding to a typical working day, where the time has been discretized in 5 buckets.

For each user, a set of FSM are incrementally created in an unsupervised way from the sequences of episodic elements that are generated at the end of each day. Given a sequence of episodic element $\{ee_1, ee_2, \ldots, ee_k\}$, the system has to decide (a) whether the sequence is an instance of an existing FSM, possibly modifying the FSM to accommodate the sequence of episodic element or (b) to create a new FSM, since the given sequence is not well matched by any existing FSM. This decision is described in the algorithm 1 and can be explained as follows:

1. Given a candidate FSM, we temporarily update it with the sequence of EEs, possibly adding new states, increasing the output value on each transition between existing states or creating new transitions with output value 1 if such transitions do not appear in the original FSM.

2. We compute the degree of match DoM between the sequence of EEs and the FSM as the product of each output value of the state transitions triggered by the sequence of EEs, normalized by N_{FSM}, that is, the number of sequences of EEs that have been matched with the FSM under evaluation.

3. If the degree of match is below a certain threshold θ, it means that the given FSM is not a good match for the sequence of EEs, otherwise the FSM is marked as possible candidate.

4. If the list of candidate FSM is empty, a new FSM is created, corresponding to the sequence of EEs.

5. If the list of candidate FSM is not empty, the sequence of EEs is matched with the FSM with the highest DoM. For all the other discarded FSMs, the temporary modifications to the states, transitions and output values are undone.

The threshold θ of the degree of match with a given FSM is defined as $\frac{1}{N_{\text{FSM}}}^{\lambda}$, where N_{FSM} is the number of sequences of EEs that have been matched with the given FSM, and $\lambda \in \{1, \ldots, k\}$ is a parameter that control how much diversity between the sequence of EEs and the FSM is accepted to consider the FSM a good match for the sequence of EEs.

Figure 3 shows the tentative match between the sequence of EEs {Home, Office, Bar, Cinema, Home} and the existing FSM of Figure 2. The output value of the transitions $s_0 \rightarrow s_1$ and $s_1 \rightarrow s_2$ is increased by 1, and the states and transitions corresponding to Office→Bar, Bar→Cinema and Cinema→Home are added to the FSM with output value 1. Assuming that $N_{\text{FSM}} = 6$, the degree of match in this case is $\frac{6}{6} \cdot \frac{6}{6} \cdot \frac{1}{6} \cdot \frac{1}{6} \cdot \frac{1}{6} = 0.0046$. If the threshold $\theta = \frac{1}{6}^{\lambda}$ is parameterised with $\lambda \in \{1, 2, 3, 4, 5\}$, the degree of match between the sequence of EEs and the FSM is above θ, thus putting the FSM into the candidate set.

By attributing sequences of EE to existing state machines or by generating new ones, we propose an incremental method

Algorithm 1 Finite State Machine Matching

```
Input: models, EEs
Output: selectedFSM
  degrees = Map(FSM, degree)
  for FSM in models do
      UPDATE(FSM, EEs)
      degrees.add COMPUTEDEGREE(FSM, EEs)
      UNDO(FSM, EEs)
  if selectedFSM != null then
      UPDATE(selectedSEM, EEs) return selectedFSM
  else
      selectedFSM = generateNewFSM(EEs)

  function UPDATE(FSM, EEs)
      previousEE = FSM.startEvent
      for EE in EEs do
          if FSM.contains(EE) then
              FSM.get(EE).incrementCounter
          else
              FSM.add(EE)
          if FSM.containsDirectedEdge(previousEE, EE)
  then
              FSM.incrementEdge previousEE, EE
          else
              FSM.createEdge previousEE, EE
          previousEE = EE

  function UNDO(FSM, EEs)
      previousEE = FSM.startEvent
      for EE in EEs do
          FSM.get(EE).decrementCounter
          FSM.decrementEdge previousEE, EE
          previousEE = EE

  function COMPUTEDEGREE(FSM, EEs)
      weight = 1
      previousEE = FSM.startEvent
      for EE in EEs do
          weight = weight * getWeight(previousEE, EE)
          previousEE = EE
      return weight/FSM.numberOfActivations
```

to effectively aggregate EEs into SEMs, each SEM being modeled by the corresponding finite state machine. The proposed model is flexible regarding the insertion of additional EEs, the deletion of EEs or the substitution of one EE by another. However, the intrinsic time-ordering of the buckets makes it very order-dependent and not suitable for unordered SEMs.

4.2 Frequent pattern model

We propose an alternative approach (FPM) for the generation of SEM based on frequent itemset mining and frequent sequence mining. Frequent itemset mining aims at finding in a list of transactions, the (unordered) sets of items which are present in more than s_i transactions, where s_i is the minimum support necessary to be defined as frequent itemset. Conversely, sequence mining aims at finding the (ordered) sequences which are present in more than s_s transactions. These methods offer the advantage of working with transactions of different sizes, making the bucketing introduced in the previous model unnecessary. A second advantage is that the combination of the two methods allow for the detection of both ordered and unordered patterns, making this approach more flexible than the finite state machines. Itemsets and sequences being two closely related concepts, we will, when possible, use the term *pattern* to refer indifferently to one or the other.

Once frequent patterns of both type have been detected, a list of boolean features is computed for each SEM based on whether the SEM matches a specific pattern or not. Finally a hierarchical clustering is performed on these binary features in order to group SEM which share similar patterns.

4.2.1 Incremental frequent itemset mining

The frequent itemset mining algorithm used in routineSense is based on the FUP algorithm [19]. Although more recent and more efficient algorithms have been developed, FUP has the advantage of being base on the Apriori algorithm which can also be altered for sequence mining. It is therefore possible to share the required scans of the database for both frequent itemset mining and sequence mining.

However, FUP as most of the incremental itemset mining algorithm uses a support s_i relative to the size of the database (e.g. $s_i = 10\%$ means that to become frequent, the itemset needs to appear in at least 10% of the transactions present in the database). This is not suitable for routineSense as it would allow itemsets which were previously frequent to become non-frequent if they don't appear in the next increments. This behavior would be equivalent to forgetting. Although forgetting is a property of the brain, it is not suitable in our case to forget. Information which may have required some user input can be deleted and, at a later time, the user would be prompted again for the same information because the itemset would have been frequent again. To this purpose, FUP is adapted for absolute support(e.g. $s_i = 7$ means that to become frequent, the itemset needs to appear in at least 7 transactions present in the database).

We define a database $\mathcal{DB} : [day_1, day_2, ..., day_n]$, where $day_i = [ee_{i1}, ee_{i2}, ..., ee_{id}]$ represents a day i as an ordered sequence of d EEs and an increment $increment : [day_{n+1}, day_{n+2}, ..., day_m]$ following the same notation. The frequent itemsets of k-items present in $\mathcal{DB}$ before the increment are denoted as $\mathcal{L}_k$. The modified FUP algorithm, presented in Algorithm 2, is used to process a new increment and update the list of frequent itemsets. It follows the bottom-up approach of the standard Apriori algorithm, but makes use of the list of frequent itemsets discovered in previous iteration to reduce the necessary computation performed at each scan.

The modified FUP algorithm will generate candidate itemsets of size $k + 1$ using the method presented in algorithm 3. This method makes use of the frequent itemsets of size k and applies the same principle as the original Apriori algorithm: a k-itemset can be frequent only if all of its subsets are frequent. The modified FUP algorithm will then test these candidates against the database and retain only the frequent candidates. This process is repeated recursively until the set of retained frequent candidates is empty.

4.2.2 Incremental frequent sequence mining

In parallel to frequent itemset mining, we use frequent sequence mining to be able to capture the temporal rela-

Algorithm 2 Frequent itemset mining

Input: $\mathcal{DB}, increment, Set < \mathcal{L}_k >$
Output: $Set < \mathcal{L}_k >$ //updated
  update of $\mathcal{L}_1$
  for day in *increment* **do**
    EEs = getUniqueEEs(day)
    if $X \in EEs$ in $\mathcal{L}_1$ **then**
      X.support++
    else
      $Candidate_1 = Candidate_1 \cup X$
  for day in $\mathcal{DB}$ **do**
    if $support(X) \in \mathcal{C}_1$ is subset of day **then**
      X.support++
  if $support(X) \in \mathcal{C}_1 > s_i$ **then**
    $\mathcal{L}_1 = \mathcal{L}_1 \cup X$

  If $|\mathcal{L}_{k-1}| > 1$ update $\mathcal{L}_k$:
  $\mathcal{C}_k$ = aprioriGen($\mathcal{L}_{k-1}$)
  $\mathcal{C}_k = \mathcal{C}_k - \mathcal{L}_k$
  for day in *increment* **do**
    if $X \in \mathcal{C}_k$ **then**
      X.support++
    if $X \in \mathcal{L}_k$ **then**
      X.support++
  for day in $\mathcal{DB}$ **do**
    if $X \in \mathcal{C}_k$ **then**
      X.support++
  if $support(X) \in \mathcal{C}_k > s_i$ **then**
    $\mathcal{L}_k = \mathcal{L}_k \cup X$

Algorithm 3 Apriori generation

Input: $\mathcal{L}_{k-1}$
Output: $\mathcal{L}_k$
  sort($X \in \mathcal{L}_k$)
  $\mathcal{L}_k = \{\}$
  for $X \in \mathcal{L}_k$ **do**
    for $Y \in \mathcal{L}_k$ **do**
      if $(X - X.last) == (Y - Y.last)$ **then**
        $\mathcal{L}_k = \mathcal{L}_k \cup \{X \cup Y\}$
  return $\mathcal{L}_k$

tionships between EEs when available. To this purpose we use the GSP algorithm [20]. This algorithm, originally not incremental, modifies the generation of the candidate itemsets of the Apriori algorithm to enforce the time-ordering of the sequence. For example $\mathcal{L}_{k-1} = \{A \rightarrow B, A \rightarrow C\}$ will lead to the generation of candidate sequences $\{A \rightarrow B \rightarrow C, A \rightarrow C \rightarrow B, A \rightarrow BC\}$. We can note that the right hand side of the last sequence is composed of simultaneous event B and C which is assumed impossible in routineSense: the user is doing only one activity at a time. The original GSP algorithm contains additional features such a definition of a minimum or maximum gap between elements of a sequence or definition of a sliding window within which the elements must be found in order to be eligible for becoming a frequent sequence. We do not need these features as they were initially intended to filter shopping basket databases, which is why these features are not retained in the context of routineSense and therefore, not implemented. Our algorithm for sequence mining is similar to the modified FUP algorithm (algorithm 2) presented previously, the difference being that the generation of the candidate sequences is done using the sequence candidates generation of GSP as defined in algorithm 4.

Algorithm 4 Sequence Apriori generation

Input: $\mathcal{L}_{k-1}$
Output: $\mathcal{L}_k$
  sort($X \in \mathcal{L}_k$)
  $\mathcal{L}_k = \{\}$
  for $X \in \mathcal{L}_k$ **do**
    for $Y \in \mathcal{L}_k$ **do**
      if $(X - X.last) == (Y - Y.last)$ **then**
        $\mathcal{L}_k = \mathcal{L}_k \cup \{(X - X.last) \rightarrow Y.last \rightarrow X.last\}$
        $\mathcal{L}_k = \mathcal{L}_k \cup \{(X - X.last) \rightarrow X.last \rightarrow Y.last\}$
  return $\mathcal{L}_k$

The choice of the FUP algorithm is therefore motivated by the fact that both frequent itemsets and frequent sequences can be discovered during the same iteration of FUP. This property allow for less scan of the database further reducing the computational cost of our approach.

4.2.3 Generation of binary features

Definition. A pattern (itemset or sequence) p is closed if there exist no superset s such that $support(p) = support(s)$.

Once frequent itemsets and sequences are discovered, each day can be attributed a list of boolean features: for each itemset and sequences denoted as patterns, if the day matches the pattern, the corresponding boolean is set to true, false otherwise. At this stage, frequent itemsets and sequences are filtered in order to retain only meaningful patterns. To this purpose patterns containing a single element are filtered out. Indeed, they just inform about the presence of an element without adding extra information. As an additional clustering, only closed frequent patterns are retained. This is to avoid duplication of information.

Indeed, following the definition, if a pattern is not closed, it contains the same amount of information as its superset with equal support. The consequence for our list of boolean features is that for each day, the boolean corresponding to the open pattern and the boolean corresponding to its superset with equal support would always have the same value, hence the redundancy in information.

4.2.4 Hierarchical clustering

Once the boolean features list has been defined for each day, they can be clustered based on this list. Indeed, days having a high number of patterns in common should be classified as belonging to the same SEM. To this purpose we use an agglomerative hierarchical clustering approach using a maximum distance between clusters as a stopping criterion.

The distance between 2 days is taken as the hamming distance between their respective boolean features. The hamming distance can be defined as the number of positions at which the element of two ordered list differ (e.g. $Hamming(01\mathbf{00}1, 01\mathbf{11}1) = 2$).

The linkage (i.e. distance between two clusters) is taken as full linkage to avoid any "chaining effect". Therefore the linkage of two clusters $A = \{sem_1, sem_2, ..., sem_k\}$ and $B = \{sem_1, sem_2, ..., sem_j\}$ is defined as $max\{a \in A, b \in B, Hamming(a, b)\}$.

We can notice that the clustering is not done incrementally. The growth of the number of frequent patterns is much slower than the growth of the database and therefore the complexity added by an incremental algorithm would not be justified.

4.2.5 Example

In this example we cover the entire pipeline of generation of SEMs using frequent pattern mining. The days of the user and their associated EEs are described in table 1. We set the minimum support threshold of the frequent itemset mining (S_i) and the minimum support threshold of the frequent sequence mining (S_s) as $s_i = s_s = 3$. The stopping criterion of the hierarchical clustering is set to $maxDistance = 1$.

Day	EEs
1	home, work, restaurant, work, home
2	home, work, restaurant, work, supermarket, home
3	home, work, sport center, work, parents in law
4	home, supermarket, sport center, restaurant, hotel

Table 1: Dataset example

The frequent itemset mining result is given in table 2. As we can see, for an itemset size k, only the frequent $(k-1)$-itemsets are used to generate the candidates.

itemset size	{itemsets}	support
1	$\{home\}$	4
	$\{work\}$	3
	$\{restaurant\}$	3
2	$\{home, work\}$	3
	$\{home, restaurant\}$	3

Table 2: Frequent itemsets

The frequent sequence mining result is given in table 3. The results are very similar to the frequent itemsets for two reasons: First, the same minimum support threshold was used for both pattern mining techniques. Secondly, the ordering of the EEs of each day is quite similar, leading to strong temporal correlation. However, we can notice that sequences involving the same EE are allowed (whereas they where prohibited for the frequent itemset mining, due to the definition of sets).

sequence size	{sequences}	support
1	$\{home\}$	4
	$\{work\}$	3
	$\{restaurant\}$	3
2	$\{home \rightarrow work\}$	3
	$\{home \rightarrow restaurant\}$	3
	$\{work \rightarrow work\}$	3
3	$\{home \rightarrow work \rightarrow work\}$	3

Table 3: Frequent sequences

Once frequent patterns are found, they are filtered by removing the 1-patterns and the open patterns. leading to the ordered list of pattern: $\{home, work\}$, $\{home, restaurant\}$, $\{home \rightarrow restaurant\}$ and $\{home \rightarrow work \rightarrow work\}$. Each day is then matched again these pattern resulting in the boolean features described in table 4.

Day	boolean features
1	1, 1, 1, 1
2	1, 1, 1, 1
3	1, 0, 0, 1
4	0, 1, 1, 0

Table 4: Boolean features ordered as: $\{home, work\}$, $\{home, restaurant\}$, $\{home \rightarrow restaurant\}$, $\{home \rightarrow work \rightarrow work\}$

Applying a hierarchical clustering of the boolean features, we obtain the following clusters representing SEMs: $\{day1, day2\}, \{day3\}, \{day4\}$.

5. EVALUATION

Both finite state machine and frequent itemsets models were evaluated against a real world dataset. The Nokia data challenge dataset [21][22] contains readings from various phone sensors for 191 users. For our study, due to the sparsity of some sensors data, we focused on the GPS traces of the users extended by the GPS location of the wifi routers in range. In order to better focus on the generation of SEMs, we take as input for our algorithms the episodic elements generated using the modified ESOINN algorithm presented by Aberer et al. in [23]. The GPS traces are clustered using the modified ESOINN and each cluster is augmented with the closest venue returned by foursquare. In order to alleviate the absence of feedback from the user, we attribute the label to each cluster once the entire dataset has been processed. This way, we avoid issues such as splitting and merging of clusters that would have required to be handled by the user. Since SEMs represent EEs with a higher level of abstraction, the evaluation of the SEMs generated using both models was not possible using solely the Nokia dataset. For this purpose, we ran a crowdsourced experiment, trusting the ability of the crowd to evaluate the SEMs. In addition, to better evaluate our approaches, we compare their results to the ones obtained with the state-of-the-art T-pattern mining algorithm.

5.1 Preprocessing

The Nokia dataset contains data about 191 users. Along with GPS information, the dataset defines also visits representing a time window during which the user was considered as being at a single location. These visits are used in order to further filter the results provided by ESOINN. However, the users were often using the phone provided for the study as a secondary phone, which was not activated all the time. Furthermore, due to low accuracy GPS reading, some signals are not usable. For this reason, we filter out users with less than at least four days with more that three EEs, reducing the number of users to 131.

We extend the dataset by adding the home location of each users during each night if this night does not contain any location information. The home location is taken as the most frequent location of the user between 10pm and 7am. The resulting days are then provided to the pattern matching algorithms. The statistics about these days are detailed in table 5.

Number of users:	preprocessing: 191 postprocessing: 131
Number of days per users:	$\mu = 33.48$, $\sigma = 33.32$
Number of EEs per day:	$\mu = 4.46$, $\sigma = 0.27$

Table 5: Statistics of the dataset. Days per users and EEs per day are defined by their average (μ) and standard deviation (σ)

Furthermore, each location record is extended with information gathered from Foursquare (i.e., location name, category, etc.) We based our evaluation on the Nokia dataset (which just contains *When?* and *Where?* information), although routineSense could leverage further semantic information about the Episodic Elements, e.g., the *What?* information. The rationale behind this choice was to test our algorithms on a large, realistic dataset that spanned over many months, at the cost of not leveraging the full potential of the algorithms. At the moment of writing this paper, we are not aware of any large dataset (i.e., hundreds of users, hundreds of days) which contains *What?*, *When?*, and *Where?*

5.2 Generation of SEM

The preprocessing phase has generated a list of days defined as sequences of EEs for each user. This list is passed on to the frequent pattern mining model and the finite state machine model. At this point, a definition of equality was needed to compare EEs. A possible formulation of equality between EEs could be based on the *type* of their venue: an EE happening in a restaurant in the city center is equal to an EE happening at the restaurant of the university. In this work we assume a stricter formulation of equality, which means that two EEs are equal if and only if their *actual venues* are equal. This is motivated by the fact that the automatic retrieval of the venues associated with the stay-points computed from the Nokia dataset was not manually validated and therefore our approach could have picked up erroneous patterns due to mislabeled venues. Strict equality prevents these patterns from being detected.

5.2.1 Finite state machine

As defined in the previous section, the finite state machine approach requires a fix number of buckets. We set this number to 24, meaning that each day contains 24 EEs. Since, the average number of EEs per day is 4.46, we need to interpolate the representation of the days. This interpolation is done in the following fashion: due to the addition of home locations in the preprocessing phase, we are sure that at least the first bucket of the day (i.e. from midnight to 1a.m.) contains a EE. From there, if the original day contains an EE happening within a bucket's time slot, this bucket will take the information of the EE, otherwise, the information of the previous bucket will be used. In case multiple EEs happen during the same bucket's time slot a majority rule is used to pick the most frequent EEs.

Once the days are interpolated, they are provided to the finite state machine model that will decide, based on the threshold θ, if the day contributes to an existing FSM or if there is need for a new FSM. We empirically set this threshold $\theta = 5$. The resulting FSMs are taken as the different type of SEMs.

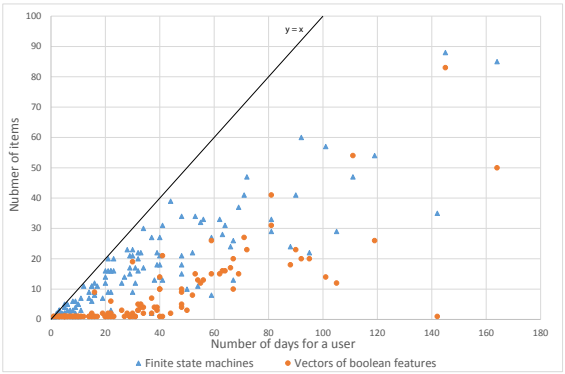

Figure 4: Number of finite state machine created and number of unique boolean features combinations as a function of the number of days per user.

5.2.2 Frequent pattern mining

To evaluate the FPM approach, the list of days is passed to the modified version of the FUP algorithm described above by weekly increments. In other words, groups of 7 days are passed to FUP until all of the days of the user are processed. The minimum support for both frequent itemsets and sequences is set to 7 (i.e. a pattern needs to appear in 7 different days before being labeled as frequent). At the end of each iteration, boolean features are generated for each of the 7 days and are placed in a set containing all the previous features lists. A hierarchical clustering with a maximum distance between clusters of 3 is then performed on this set to group days into SEMs. Although in a real application the cluster to which a day belongs can be determined on the fly when the day is accessed, for the sake of the user study the list of days is scanned one last time to attribute a SEM label to each day. The statistics of the resulting SEMs are provided in table 6.

We can note that a day with boolean features $< 0, 0, ..., 0 >$ means that it matches none of the frequent patterns. Days having such vector are therefore exceptional and must be treated separately. Indeed, these days are very different from one another and should not be used in the clustering phase, instead they are grouped in a separate cluster.

Length of the boolean features:	9.55
Number of unique boolean features:	5.52
Number of SEMs:	3.37

Table 6: Statistics (averaged per user) for the frequent pattern matching approach

The number of finite state machines generated by the FSM model and the number of unique vectors of boolean features created by the frequent pattern mining model for each user are depicted in figure 4 as a function of the number of days collected for each users. We can notice a long tail of users having only 1 vector of boolean features ($< 0, 0, ..., 0 >$). This implies that no frequent patterns were discovered for these users. This behavior is an artifact of the nokia dataset and happens essentially with users with not enough data and irregular routines: 97% of these special cases have less

than 30 days of data.

5.3 Comparison with T-patterns

In order to further evaluate the two approaches developped in this paper, we compare them to the well known T-pattern mining algorithm developped in [17]. The algorithm takes as input raw GPS traces and processes as follows: i) Superpose the GPS traces on a grid and determine the density of each cell of the grid based on the number of traces visiting the cell. ii) Define regions of interest (ROI) by grouping cells which have a density above a certain threshold δ. iii) Generate T-patterns in a bottom-up fashion, taking as parameters a minimum support (usually equal to δ) and a maximum transition time between states.

We use the open-source version of the T-pattern algorithm, and we choose its parameters to match the conditions of both our approaches: No interpolation between two points of a trace, an absolute minimum support and density threshold $\delta = 7$, a grid subdivision of $100 * 100$ and a time limite infinite. There is therefore no time constraint for the generation of T-patterns.

5.4 Crowdsourced evaluation

We run a user study on Amazon Mechanical Turk to assess if the routines identified by T-patterns, FPM and FSM are deemed reasonable by humans. To this extent, we designed an experiment where each worker was presented with 4 different sets of daily activities, and 6 different choices. The choices were the following: (*Outlier*) 4 buttons for each single day, to signal that the given set of activities was an outlier among the other 3; (*All-match*) 1 button to signal that all the 4 days could be mapped to the same routine; (*No-match*) 1 button to signal that all the 4 days were different (therefore not mappable to the same routine). All the tasks have been generated automatically, in a randomized fashion. In total, combining the 3 different approaches, more than 3800 tasks (paid $0.02 each) were performed by 279 crowd workers.

Table 7 reports the results we obtained from the crowd, with FPM as a clear winner. Given the 6 choices presented to the user, on the same task a randomized baseline algorithm would perform with a 16.7% accuracy. Realistically though, an algorithm that performs poorly could obtain way worse results than the randomized baseline in an evaluation with human subjects, e.g., the algorithm could fail to identify many important routines (therefore classifying most of the days as *No-match*), while they will not go unnoticed to the crowd workers.

It is also interesting to notice that a state-of-the-art algorithm like T-patterns performs almost as the randomized baseline for this specific task, confirming that routineSense is tackling an hard (and novel) variant of a well-known class of problems. Such performance does not question the validity of T-patterns per se, but only its suitability in the context of identifying user routines.

The accuracy has been measured accordingly to the confusion matrix in Table 8.

For the sake of readability, we compacted the *Outlier* 4−categories classification task as a single entry in the matrix. The first cell contains in bold the percentage of outliers that have been correctly identified among all the tasks submitted to the workers, while the value in parenthesis represents the mislabeled outliers. All the other cells follow the usual behavior of a confusion matrix, with the diagonal reporting the accurate classifications, and all the other cells reporting the misclassifications.

	Accuracy	Sensitivity	Specificity
T-patterns	21.6%	40.6%	8.8%
FSM	44.0%	59.1%	49.7%
FPM	56.3%	74.3%	51.9%

Table 7: Crowdsourced evaluation of T-patterns, FSM and FPM (i.e., user agreement)

	Outlier	All-match	No-match
Outlier	**32.1**/(20.3)	12.7	3.8
All-match	10.3	**7.0**	1.1
No-match	0.8	0.7	**11.2**

Table 8: Confusion matrix for the classification tasks performed by the crowd (compound of T-patterns, FSM and FPM). All numbers are percentages, and the correct classifications are reported in bold. Rows are expected answers, columns are the input from the crowd.

5.5 Performance evaluation

In order to assess that our approach is suitable for nowadays devices, we also evaluated the runtime performance of both SEM generation methods. We used a Google Nexus 4 with a quadcore CPU (1.5 GHz), 2GB of RAM and the operating system Android 4.4.

Both finite state machine and frequent pattern mining approaches are compared to a baseline defined as a non-incremental version of the frequent pattern mining process, in which frequent itemsets are discovered using the original Apriori algorithm, and frequent sequences are discovered using the original GSP algorithm. The rest of the process (i.e., boolean features generation and hierarchical clustering) remain as described in the previous section.

The results presented in figure 5 show that, as weeks accumulate, the processing time required by the non-incremental method outgrows significantly our two approaches, justifying the incremental processing design choice of routineSense. The time taken to process a single week of data is below 10 seconds for both our algorithms, therefore an app that embeds routineSense will not make the phone unresponsive, nor it will hinder the user experience.

Given that the battery consumption is influenced by the CPU usage, it is also worth to notice that the incremental processing capabilities of routineSense make it a battery-conscious framework – an especially desirable feature, considering the plethora of energy-hungry applications that run on modern smartphones.

6. CONCLUSIONS AND FUTURE WORK

The framework routineSense presented in this paper endeavours to model, in an energy efficient manner, the past activities of a user, similarly to the way the human brain builds memories. We propose two concurrent methods to aggregate low level pieces of memories (EE) into more abstract simple episodic memories. To compensate for the lack of information about SEM in the dataset used for the experi-

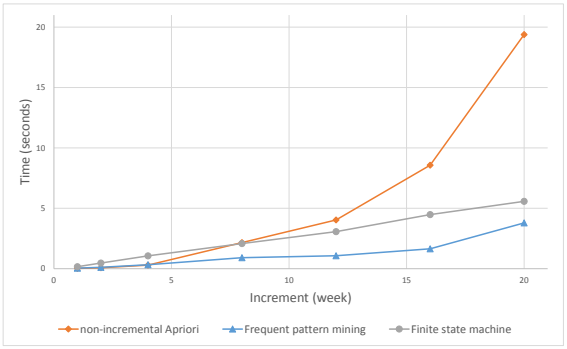

Figure 5: Time required to process a new week of data for both SEM generation methods, compared to a non-incremental method based on original Apriori and GSP.

ment, we conducted a crowd-sourced experiment to evaluate the soundness of our approach. The results show that we improve almost 3 times on the current state-of-the-art, without leveraging any user feedback. The performance evaluation also shows that both our methods for generating SEMs are suitable for mobile computation.

As a future work, we want to lift the assumptions that the length of an SEM should be fixed to a day, enabling us to cover even longer SEMs. A second goal as future work regards the collection of a dataset that better represents the type of data handled by routineSense. Finally, the possibilities offered by the semantic approach taken in routineSense are not fully explored. Although the data collected by the routineSense are properly structured, additional research on semantic reasoning can be exploited to improve the capabilities of the framework.

7. ACKNOWLEDGMENTS

The MemorySense project was sponsored by Samsung Research America, and by Haslerstiftung in the context of the Smart World 11005 (MEM0R1ES) project. Portion of the research in this paper used the MDC Database (owned by Nokia and made available by the Idiap Research Institute, Switzerland).

8. REFERENCES

[1] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell, "The jigsaw continuous sensing engine for mobile phone applications," in *SenSys*, 2010.

[2] M. A. Conway, "Episodic memories," *Neuropsychologia*, 2009.

[3] Y. Zheng, L. Liu, L. Wang, and X. Xie, "Learning transportation mode from raw gps data for geographic applications on the web," in *WWW '08*, pp. 247–256, ACM, 2008.

[4] Y. Zheng, L. Wang, R. Zhang, X. Xie, and W.-Y. Ma, "Geolife: Managing and understanding your past life over maps," in *Proceedings of the The Ninth International Conference on Mobile Data Management*, IEEE Computer Society, 2008.

[5] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava, "Using mobile phones to determine transportation modes," *ACM Transactions on Sensor Networks (TOSN)*, 2010.

[6] Z. Yan, V. Subbaraju, D. Chakraborty, A. Misra, and K. Aberer, "Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach," in *ISWC '12*, pp. 17–24, Ieee, 2012.

[7] T. Choudhury, S. Consolvo, B. Harrison, J. Hightower, A. LaMarca, L. Legrand, A. Rahimi, A. Rea, G. Bordello, B. Hemingway, P. Klasnja, K. Koscher, J. A. Landay, J. Lester, D. Wyatt, and D. Haehnel, "The mobile sensing platform: An embedded activity recognition system," *PerCom '08*, 2008.

[8] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," *ACM SigKDD Explorations Newsletter*, vol. 12, no. 2, pp. 74–82, 2011.

[9] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell, "Sound-sense: Scalable sound sensing for people-centric applications on mobile phones," in *MobiSys*, 2009.

[10] N. Bicocchi, G. Castelli, M. Mamei, A. Rosi, and F. Zambonelli, "Supporting location-aware services for mobile users with the whereabouts diary," in *MOBILWARE '08*, 2008.

[11] D. Feldman, A. Sugaya, C. Sung, and D. Rus, "idiary: From gps signals to a text-searchable diary," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, p. 6, ACM, 2013.

[12] B. Guo, D. Zhang, D. Yang, Z. Yu, and X. Zhou, "Enhancing memory recall via an intelligent social contact management system," *Human-Machine Systems, IEEE Transactions on*, vol. 44, no. 1, pp. 78–91, 2014.

[13] B. Chikhaoui, S. Wang, and H. Pigot, "Adr-splda: Activity discovery and recognition by combining sequential patterns and latent dirichlet allocation," *Pervasive and Mobile Computing*, 2012.

[14] U. Blanke and B. Schiele, "Daily routine recognition through activity spotting," in *Proceedings of the 4th International Symposium on Location and Context Awareness*, pp. 192–206, Springer-Verlag, 2009.

[15] K. Farrahi and D. Gatica-Perez, "Discovering routines from large-scale human locations using probabilistic topic models," *ACM Trans. Intell. Syst. Technol.*, vol. 2, pp. 3:1–3:27, Jan. 2011.

[16] N. Eagle and A. S. Pentland, "Eigenbehaviors: Identifying structure in routine," *Behavioral Ecology and Sociobiology*, vol. 63, no. 7, pp. 1057–1066, 2009.

[17] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi, "Trajectory pattern mining," in *SIGKDD '07*, KDD '07, (New York, NY, USA), pp. 330–339, ACM, 2007.

[18] S. Furao, T. Ogura, and O. Hasegawa, "An enhanced self-organizing incremental neural network for online unsupervised learning," *Neural Networks*, vol. 20, 2007.

[19] P. S. Tsai, C.-C. Lee, and A. L. Chen, "An efficient approach for incremental association rule mining," in *Methodologies for Knowledge Discovery and Data Mining*, Springer, 1999.

[20] R. Srikant and R. Agrawal, *Mining sequential patterns: Generalizations and performance improvements*. Springer, 1996.

[21] N. Kiukkonen, J. Blom, O. Dousse, D. Gatica-Perez, and J. Laurila, "Towards rich mobile phone datasets: Lausanne data collection campaign," *ICPS*, 2010.

[22] J. K. Laurila, D. Gatica-Perez, I. Aad, J. Blom, O. Bornet, T.-M.-T. Do, O. Dousse, J. Eberle, and M. Miettinen, "The mobile data challenge: Big data for mobile computing research," in *Proceedings of the Workshop on the Nokia Mobile Data Challenge*, 2012.

[23] K. Aberer, M. Catasta, H. Radu, J.-E. Ranvier, M. Vasirani, and Z. Yan, "Memorysense: Reconstructing and ranking user memories on mobile devices," in *PERCOM Workshops '14*, IEEE, 2014.