Int J Wireless Inf Networks (2011) 18:88–99
DOI 10.1007/s10776-011-0142-4

# An Adaptive Machine Learning Algorithm for Location Prediction

**Theodoros Anagnostopoulos · Christos Anagnostopoulos · Stathes Hadjiefthymiades**

Received: 18 August 2009 / Accepted: 9 April 2011 / Published online: 23 April 2011


**Abstract** Context-awareness is viewed as one of the most important aspects in the emerging pervasive computing paradigm. Mobile context-aware applications are required to sense and react to changing environment conditions. Such applications, usually, need to recognize, classify and predict context in order to act efficiently, beforehand, for the benefit of the user. In this paper, we propose a novel adaptive mobility prediction algorithm, which deals with location context representation and trajectory prediction of moving users. Machine Learning (ML) is used for trajectory classification. Our algorithm adopts spatial and temporal on-line clustering, and relies on Adaptive Resonance Theory (ART) for trajectory prediction. The proposed algorithm applies a Hausdorff-like distance over the extracted trajectories handling location prediction. Since our approach is time-sensitive, the Hausdorff distance is considered more advantageous than a simple Euclidean norm. Two learning methods (non-reinforcement and reinforcement learning) are presented and evaluated. Finally, we compare our algorithm with Offline *k*Means and Online *k*Means algorithms. Our findings are very promising for the use of the proposed algorithm in mobile context aware applications.

T. Anagnostopoulos (✉) · S. Hadjiefthymiades
Pervasive Computing Research Group, Communication Networks Laboratory, Department of Informatics and Telecommunications, University of Athens, Panepistimiopolis, Ilissia, 15784 Athens, Greece
e-mail: thanag@di.uoa.gr

S. Hadjiefthymiades
e-mail: shadj@di.uoa.gr

C. Anagnostopoulos
Department of Informatics, Ionian University, 49100 Corfu, Greece
e-mail: christos@ionio.gr

**Keywords** Context-awareness · Location prediction · Machine Learning · Online clustering and classification · Adaptive Resonance Theory

## 1 Introduction

In order to render mobile context-aware applications intelligent enough to support users everywhere and anytime, information on the present *context* of the users has to be captured and processed accordingly. A well-known definition of context is the following: "*context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the integration between a user and an application, including the user and the application themselves*" [1]. Context refers to the current values of specific ingredients that represent the activity of an entity, situation and environmental state e.g., location, time, walking, attendance of a meeting, driving a car, traveling.

One of the more intuitive capabilities of the mobile context-aware applications is their *proactivity*. Predicting user activities and contextual ingredients, in general, enables a new class of applications to be developed along with the improvement of existing ones. The most important ingredient for mobile context-aware applications is the location of the mobile user. Estimating and predicting the future location of a mobile user enables the development of innovative, location-based services and applications [2, 12]. For instance, location prediction can be exploited for improving resource reservation in wireless networks. This facilitates the provision of location-based services by preparing and feeding them with appropriate information in advance.

Location prediction is quite similar to information prediction through classification. Specifically, ML is *the study of algorithms that improve automatically through experience*. ML provides algorithms for learning a system to:

- cluster pre-existing knowledge,
- classify observations,
- predict unknown situations based on a history of patterns, and,
- adapt to situation changes.

In addition, ML provides solutions suitable for location prediction. Context-aware applications have a set of pivotal requirements e.g., flexibility and adaptation. These requirements would strongly benefit if the learning and prediction processes could be performed in real time. We argue that the most appropriate solutions for location prediction are offline and online clustering and classification. Offline clustering is performed through the Offline *k*Means algorithm. Online clustering is accomplished through the Online *k*Means and Adaptive Resonance Theory (ART). Offline learners typically perform complete model building, which can be very costly, if the amount of samples rises. On the other hand, online learning algorithms are able to (i) detect changes in patterns and (ii) update only parts of the model. The later provides adaptation of the model. Both forms of algorithms extract a subset of patterns, i.e., clusters in a knowledge base, from an initial dataset i.e., a database of user trajectories.

Online learning is more suited for the task of prediction through classification than offline learning. That is because, in real life, user movement data often needs to be processed in an online manner, each time after a new portion of the data arrives. This is caused by the fact that such data is organized in the form of a data stream (e.g., a sequence of time-stamped visited locations).

Classification involves the matching of an unseen pattern with existing clusters in the knowledge base. We rely on a Hausdorff-like distance [5] for matching unseen trajectories to clusters. Such metric applies to convex patterns and is considered ideal for user trajectories. Hence, location prediction boils down to location classification w.r.t. Hausdorff-like distance.

In this paper, we propose an adaptive ML algorithm that

(i) clusters the user movements,
(ii) identifies changes in the user movements
(iii) adapts its knowledge structure to such changes, and,
(iv) predicts the future user location.

Specifically, we introduce two *training methods* for training our algorithm: (i) the "nearly" *zero-knowledge* and (ii) the *supervised* method. In the first method, our algorithm is incrementally trained starting with a little knowledge on the user mobility behavior. In the second method, sets of known trajectories are fed to our algorithm. In addition, we introduce two *learning methods* for the proposed algorithm regarding the success of location prediction: (i) the *non-reinforcement learning* (nRL) and (ii) the *reinforcement learning* (RL). In the nRL method, a misclassified trajectory is no further used in the model-training phase. Hence, the algorithm is no longer aware of unsuccessful predictions. In the RL method, a misclassified trajectory is introduced into the knowledge base, thus, updating appropriately the model.

We evaluate the performance of our algorithm against real users' movement. We establish important metrics for the performance assessment process w.r.t. (i) low system-requirements, i.e., storage capacity, and (ii) the required effort for model building, i.e., processing power. We assess the prediction accuracy of our algorithm, i.e., the precision of location predictions, along with the required size of the derived knowledge base. This size indicates the portion of the produced clusters out of the volume of the training patterns. Surely, we need to keep storage capacity as low as possible while maintaining good prediction accuracy. We also study the capability of our algorithm in adapting the derived model to unseen patterns. *Adaptivity* indicates the capability of the proposed algorithm to detect and update appropriately a specific part of the trained model. The algorithm should rapidly detect changes in the behavior of the mobile user and adapt accordingly through model updates. This is, however, achieved often at the expense of the prediction accuracy.

The paper is structured as follows: In Sect. 2, we present the Offline *k*Means, Online *k*Means and ART algorithms. Section 3 elaborates on location and trajectory representation. Section 4 presents the proposed algorithm based on ART. The performance assessment is presented in Sect. 5. In Sect. 6, we compare our algorithm with Offline *k*Means and Online *k*Means. Prior work is discussed in Sect. 7, while Sect. 8 concludes the paper.

## 2 Machine Learning Models

We briefly discuss the clustering algorithms adopted for our algorithm. We distinguish between offline and online clustering and elaborate on the Offline *k*Means and Online *k*Means and ART.

### 2.1 Offline *k*Means

In Offline *k*Means [3] we assume that there are $k > 1$ initial clusters of data. The objective of this algorithm is to minimize the reconstruction error, which is the total Euclidean distance between the patterns $\mathbf{u}_i$ and their representation, i.e., the cluster centers (clusters), $\mathbf{c}_i$. The reconstruction error is defined as follows:

$$E\left(\{\mathbf{c}_i\}_{i=1}^{k}|U\right) = \frac{1}{2}\sum_t \sum_i b_{i,t}||\mathbf{u}_t - \mathbf{c}_i||^2 \quad (1)$$

where

$$b_{i,t} = \begin{cases} 1, \textit{ if } ||\mathbf{u}_t - \mathbf{c}_i|| = \min_l ||\mathbf{u}_t - \mathbf{c}_l|| \\ 0, \textit{ otherwise} \end{cases},$$

$U = \{\mathbf{u}_t\}$ is the total set of patterns and $C = \{\mathbf{c}_i\}$, $i = 1,\ldots, k$ is the set of clusters. $b_{i,t}$ is 1 if $\mathbf{c}_i$ is the closest center to $\mathbf{u}_t$ in Euclidean distance. For each incoming $\mathbf{u}_t$ each $\mathbf{c}_i$ is updated as follows:

$$\mathbf{c}_i = \frac{\sum_t b_{i,t}\mathbf{u}_t}{\sum_t b_{i,t}} \quad (2)$$

Since the algorithm operates in offline mode, the initial clusters can be set during the training phase and cannot be changed, for instance increased or relocated, during the testing phase.

### 2.2 Online *k*Means

In Online *k*Means [3] we assume that there are $k > 1$ initial clusters that split the data. Such algorithm processes unseen patterns one by one and performs *small* updates in the position of the appropriate cluster ($\mathbf{c}_i$) at each step. The algorithm does not require a training phase. The update for each new (unseen) pattern $\mathbf{u}_t$ is the following:

$$\mathbf{c}_i = \mathbf{c}_i + \eta \cdot b_{i,t} \cdot (\mathbf{u}_t - \mathbf{c}_i)$$

This update moves the *closest* cluster (for which $b_{i,t} = 1$) toward the input pattern $\mathbf{u}_t$ by a factor of $\eta$. The clusters that are found at bigger distances from the considered pattern are not updated. The semantics of $b_{i,t}$, $\eta$ and $(\mathbf{u}_t - \mathbf{c}_i)$ are:

$b_{i,t} \in \{0, 1\}$ denotes which cluster is being modified, $\eta \in [0, 1]$ denotes how much is the cluster shifted toward the new pattern, and, $(\mathbf{u}_t - \mathbf{c}_i)$ denotes the distance to be learned.

Since the algorithm is online, the initial clusters should be known beforehand[1] and can only be relocated during the testing phase. The number of clusters remains constant. Therefore, the algorithm exhibits limited flexibility.

### 2.3 Adaptive Resonance Theory

The ART algorithm [4] is an online learning scheme, in which the set of patterns $U$ is not available during training. The patterns are received one by one and the model is updated progressively. The term *competitive learning* is used for ART denoting that the local clusters *compete* among themselves to assume the "responsibility" for representing an unseen pattern. The model is also called *winner-takes-all* because one cluster "wins the competition" and gets updated, and the others are not updated at all.

The ART approach is *incremental*. This means that one starts with one cluster and adds a new one, if needed. Given an input $\mathbf{u}_t$, the distance $b_t$ is calculated for all clusters $\mathbf{c}_i$, $i = 1,.., k$, and the closest (e.g., minimum Euclidean distance) to $\mathbf{u}_t$ is updated. Specifically, if the minimum distance $b_t$ is smaller than a certain threshold value, named the *vigilance*, $\rho$, the update is performed as in Online *k*Means (see Eq. 3). Otherwise, a new center $\mathbf{c}_{k+1}$ representing the corresponding input $\mathbf{u}_t$ is added in the model (see Eq. 3). It is worth noting that the vigilance threshold refers to the criterion of considering two patterns equivalent or not during the learning phase of the algorithm. As it will be shown, the value of vigilance is considered essential in obtaining high values of corrected classified patterns. The following equations are adopted in each update step of ART:

$$b_t = ||\mathbf{c}_i - \mathbf{u}_t|| = \min_{l=1}^{k} ||\mathbf{c}_l - \mathbf{u}_t||$$
$$\begin{cases} \mathbf{c}_{k+1} \leftarrow \mathbf{u}_t & \textit{if } b_t > \rho \\ \mathbf{c}_i = \mathbf{c}_i + \eta(\mathbf{u}_t - \mathbf{c}_i) & \textit{otherwise} \end{cases} \quad (3)$$

## 3 Location and Trajectory Representation

Several approaches have been proposed in order to represent the movement history, or trajectory, of a mobile user [15]. We adopt a spatiotemporal history model in which the trajectory is represented as the sequence of 3-D points (3DPs) visited by the moving user, i.e., time-stamped trajectory points in a 2D surface. The spatial attributes in that model denote latitude and longitude.

Let $\mathbf{e} = (x, y, t)$ be a 3DP. The *user trajectory* $\mathbf{u}$ consists of several time-ordered 3DPs, $\mathbf{u} = [\mathbf{e}_i] = [\mathbf{e}_1, \ldots, \mathbf{e}_N]$, $i = 1, \ldots, N$ and is stored in the system's database. It holds that $t(\mathbf{e}_1) < t(\mathbf{e}_2) < \ldots < t(\mathbf{e}_N)$, i.e., time-stamped coordinates. The $x$ and $y$ dimensions denote the latitude and the longitude while $t$ denotes the time dimension ($t(\cdot)$ returns the time coordinate of $\mathbf{e}$). Time assumes values between 00:00 and 23:59. To avoid state information explosion, trajectories contain time-stamped points sampled at specific time instances. Specifically, we sample the movement of each user at $1.66 \times 10^{-3}$ Hertz (i.e., once every 10 min). Sampling at very high rates (e.g., in the order of a Hertz) is meaningless, as the derived points will be highly correlated.

In our algorithm, $\mathbf{u}$ is a finite sequence of $N$ 3DPs, i.e., $\mathbf{u}$ is a $3{\cdot}N$ dimension vector. We have adopted a value of $N = 6$ for our experiments. This denotes that we estimate

[1] One possible approach to determine the initial $k$ clusters is to select the first $k$ distinct instances of the input sample $U$.

the future position of a mobile terminal from a trajectory of 50 min (i.e., 5 samples). Specifically, we aim to query the system with a $N$-1 3DP sequence so that our algorithm returns a 3DP, which is the predicted location of the mobile terminal.

A *cluster trajectory* $\mathbf{c}$ consists of a finite number of 3DPs, $\mathbf{c} = [\mathbf{e}_i]$, $i = 1, \ldots, N$ stored in the knowledge base. Note that a cluster trajectory $\mathbf{c}$ and a user trajectory $\mathbf{u}$ are vectors of the same length $N$. This is because $\mathbf{c}$, which is created from ART based on unseen user trajectories, is a representative itinerary of the user movements. In addition, the *query trajectory* $\mathbf{q}$ consists of a number of 3DPs, $\mathbf{q} = [\mathbf{e}_j]$, $j = 1, \ldots, N-1$. It is worth noting that $\mathbf{q}$ is a sequence of $N-1$ 3DPs. Given a $\mathbf{q}$ with a $N-1$ history of 3DPs we predict the $\mathbf{e}_N$ of the closest $\mathbf{c}$ as the next user movement.

## 4 Mobility Prediction Algorithm

From the ML perspective the discussed location prediction problem refers to an $m + l$ model [13]. In $m + l$ models we have $m$ steps of user trajectory and we want to predict the future user movement after $l$ steps (the steps have time-stamped coordinates). In our case, $m = N-1$, i.e., the query trajectory $\mathbf{q}$, while $l = 1$, i.e., the predicted $\mathbf{e}_N$. We develop a new spatiotemporal algorithm (C) which given $\mathbf{q}$ can predict $\mathbf{e}_N$. Specifically, $\mathbf{q}$ and $\mathbf{c}$ are trajectories of different length–dimension—thus we use a Hausdorff-like measure for calculating the distance $\|\mathbf{q}-\mathbf{c}\|$. Given query $\mathbf{q}$, the proposed algorithm C attempts to find the nearest cluster $\mathbf{c}$ in the knowledge base and, then, take $\mathbf{e}_N$ as the *predicted* 3DP. For evaluating C, we compute the Euclidean distance between the *predicted* 3DP and the *actual* 3DP (i.e., the real user movement). If such distance is greater than a preset error threshold $\theta$ then prediction is not successful. After predicting the future location of a mobile terminal, the C algorithm may receive feedback from the environment considering whether the prediction was successful or not, and reorganize the knowledge base accordingly [14]. In our case, the feedback is the actual 3DP observed in the terminal's movement. We can have two basic versions of the C algorithm:

1. the C-RL algorithm, which reacts with the environment and learns new patterns through reinforcement learning once an unsuccessful prediction takes place, and,
2. the C-nRL algorithm, which is unaware of unsuccessful predictions.

Specifically,

in case of an unsuccessful prediction, the C-RL appends the actual 3DP to $\mathbf{q}$ and reinforces such extended sequence in the model considering as new knowledge, i.e., an unseen user movement behavior.

in the case of a successful prediction, we do not reinforce C to learn. A successful prediction refers to a well-established prediction model for handling unseen user trajectories.

The heart of the proposed algorithm is the ART algorithm. ART gathers unseen user trajectories to existing cluster trajectories or creates new cluster trajectories depending on the vigilance value. ART is taking the $\mathbf{u}_1$ pattern from the incoming set $U$ of patterns and stores it as the $\mathbf{c}_1$ cluster in the knowledge base. For the $t$-th unseen user trajectory the following procedure is followed (see Table 1): The algorithm computes the Euclidean distance $b_t$ between $\mathbf{u}_t$ and the closest $\mathbf{c}_i$. If $b_t$ is smaller than the vigilance $\rho$ then $\mathbf{c}_i$ is updated from $\mathbf{u}_t$ by the $\eta$ factor. Otherwise, a new cluster $\mathbf{c}_j \equiv \mathbf{u}_t$ is inserted into the knowledge base. The ART algorithm is presented in Table 1.

Let $T$, $P$ be subsets of $U$ for which it holds that $T \subseteq P \subseteq U$. The $T$ set of patterns is used for training C, that is, C develops a knowledge base corresponding to the supervised training method. The $P$ set is used for performing on-line predictions. We introduce the C-RLT and the C-nRLT versions, which are the C-RL and C-nRL algorithms trained with the $T$ set, respectively. In addition, once the $T$ set is null then C is not trained beforehand corresponding to the zero-knowledge training method and performs on-line prediction with the set $P$. In this case, we get the C-RLnT and the C-nRLnT corresponding to the C-RL and C-nRL, respectively, when the training phase is foreseen.

Moreover, in order for the C algorithm to achieve prediction, an approximate Hausdorff-like metric [5] is adopted to estimate the distance between $\mathbf{q}$ and $\mathbf{c}$. Specifically, the adopted formula calculates the point-to-vector distance between $\mathbf{e}_j \in \mathbf{q}$ and $\mathbf{c}$, $\delta'(\mathbf{e}_j, \mathbf{c})$, as follows:

**Table 1** The adopted ART for the C algorithm

| | |
|---|---|
| 1. | $j \leftarrow 1$ |
| 2. | $\mathbf{c}_j \leftarrow \mathbf{u}_j$ |
| 3. | **For** ($\mathbf{u}_t \in U$) **Do** |
| 4. | $b_t = \|\mathbf{c}_j - \mathbf{u}_t\| = \min_{l=1,\ldots,j}\|\mathbf{c}_l - \mathbf{u}_t\|$ |
| 5. | **If** ($b_t > \rho$) **Then**/*expand knowledge*/ |
| 6. | $j \leftarrow j + 1$ |
| | $\mathbf{c}_j \leftarrow \mathbf{u}_t$ |
| 7. | **Else** |
| 8. | $\mathbf{c}_j \leftarrow \mathbf{c}_j + \eta(\mathbf{u}_t - \mathbf{c}_j)$/*update model locally*/ |
| 9. | **End If** |
| 10. | **End for** |

$$\delta'(\mathbf{e}_j, \mathbf{c}) = ||\mathbf{f}_i - \mathbf{e}_j||_{\min_{\mathbf{f}_i} |t(\mathbf{f}_i) - t(\mathbf{e}_j)|}$$

where $\|\cdot\|$ is the Euclidean norm for $\mathbf{f}_i \in \mathbf{c}$ and $\mathbf{e}_j$. The $\delta'(\mathbf{e}_j, \mathbf{c})$ value indicates the minimum distance between $\mathbf{e}_j$ and $\mathbf{f}_i$ w.r.t. the time stamped information of the user itinerary, that is the Euclidean distance of the closest 3DPs in time. Hence, the overall distance between the $N-1$ in length $\mathbf{q}$ and the $N$ in length $\mathbf{c}$ is calculated as

$$\delta_{N-1}(\mathbf{q}, \mathbf{c}) = \frac{1}{N-1} \sum_{\mathbf{e}_j \in \mathbf{q}} \delta'(\mathbf{e}_j, \mathbf{c}) \quad (4)$$

Figure 1 depicts the process of predicting the next user movement considering the diverse versions of the proposed C algorithm. Specifically, once a query trajectory $\mathbf{q}$ arrives, then C attempts to classify $\mathbf{q}$ into a known $\mathbf{c}_i$ in the knowledge base w.r.t. Hausdorff metric. The C algorithm returns the predicted $\mathbf{e}_N \in \mathbf{c}_i$ of the closest $\mathbf{c}_i$ to $\mathbf{q}$. Once such result refers to an unsuccessful prediction w.r.t. a preset error threshold $\theta$ then the C-RLT (or the C-RLnT) extend the $\mathbf{q}$ vector with the actual 3DP and insert $\mathbf{q}$ into the knowledge base for further learning according to the algorithm in Table 1 (feedback). By adopting the non-reinforcement learning method, the C (the C-nRLT/-nRLnT versions) does not give feedback to the knowledge base (no adaptation).

## 5 Adaptation and Location Prediction Evaluation

We evaluated our adaptive model in order to assess its performance. In our experiments, the overall real user

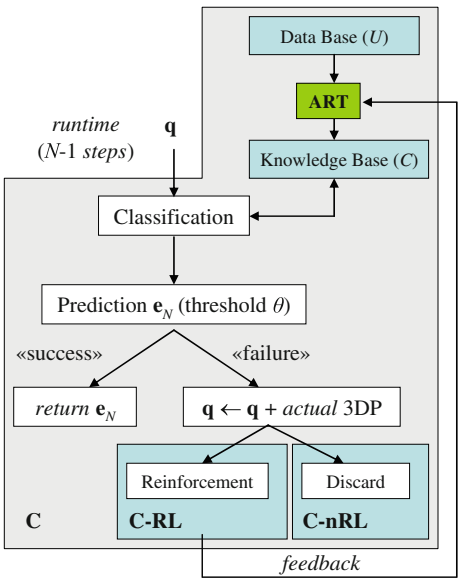


**Fig. 1** The proposed adaptive algorithm C for knowledge adaptation and location prediction

movement space has a surface of 540 km$^2$. Such space derives from real GPS trace captured in Denmark [6]. The GPS trace was fed into our model and the performance of the C system w.r.t. predefined metrics was monitored. Table 2 indicates the parameters used in our experiments. The values of the parameters refer to the real GPS data set.

The GPS traces including 1,200 patterns were preprocessed. We produced two training files and two test files as depicted in Fig. 2. The first training file, *TrainA*, is produced from the first half of the GPS trace records. The second training file, *TrainB*, consists of a single trace record. The first test file, *TestA*, is produced from the entire set of trace records, including -in ascending order- the first half of the GPS traces and the other half of unseen traces. Finally the second test file, *TestB*, is produced from the entire set of the GPS trace records, including—in ascending order—the second half of unseen traces and the first half of the GPS traces. During the generation of the training and test files, white noise was artificially induced into the trace records.

We analyzed the user movement and found that the user moves from *TownA* to *TownB* either directly (i.e., one pattern) or indirectly through *TownC* (i.e., new pattern). These patterns are stored in the first 600 patterns of *TestA*. During time the user changes behavior and she or he discovers a third indirect way of going from *TownA* to *TownB* via *TownD* (i.e., newer pattern). This pattern is stored in the second 600 patterns of *TestA*. We prove that the proposed algorithm discovers such changes of the user behavior and adapt its knowledge base accordingly.

We have to quantitatively and qualitatively evaluate the proposed algorithm. We introduce the following quantitative and qualitative parameters:

(a) the precision achieved by the prediction scheme. Specifically the higher the precision the more accurate the decisions on the future user location
(b) the size of the underlying knowledge base. We should adopt solutions with the lowest possible knowledge base size. Such solutions are far more efficient and feasible in terms of implementation.
(c) the capability of the algorithm to rapidly react to changes in the movement pattern of the user/mobile terminal and re-adapt.

We define *precision*, $p$, as the fraction of the correctly predicted locations, $p_+$, against the total number of predictions made by the C system, $p_{\text{total}}$, that is,

$$p = \frac{p_+}{p_{\text{total}}}$$

In the following sub-sections, we evaluate the diverse versions of C w.r.t. training and learning methods by examining the algorithm convergence (speed of learning

**Table 2** Experimental parameters

| Parameter | Value | Comment |
|---|---|---|
| Learning rate ($n$) | 0.5 | In case of a new pattern $\mathbf{u}_t$, the closest cluster $\mathbf{c}_i$ is moved toward $\mathbf{u}_t$ by half the spatial and temporal distance. |
| Spatial coefficient of vigilance ($\rho_s$) | 100 m | Two 2D points are considered different if their spatial distance exceeds 100 m. |
| Temporal coefficient of vigilance ($\rho_t$) | 10 min | Two time-stamps are considered different if their temporal distance exceeds 10 min. |
| Precision threshold/location accuracy ($\theta$) | 10 m | The predicted location falls within a circle of radius 10 m from the actual location. |

Such accuracy level is considered appropriate for the kind of applications where location prediction can be applied (see Sect. 1 or [12])

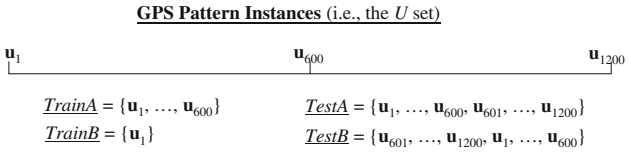


**Fig. 2** The generated GPS trace files for experimentation

and adaptation) and the derived precision on prediction future locations.

## 5.1 Convergence of C-RLT and C-RLnT

The C algorithm converges once the knowledge base does not expand with unseen patterns, i.e., the set $U$ does not evolve. In Fig. 3, we plot the number of the clusters $|U|$ that are generated from the C-RLT/-RLnT models with reinforcement learning during the testing phase. The horizontal axis denotes the incoming (time-ordered) GPS patterns. The point (.) marked line depicts the behavior of the C-RLT-1 model trained with *TrainA* and tested with *TestA*. In the training phase, the first 600 patterns of *TrainA* have gradually generated 70 clusters in $U$. In the testing phase, the first 600 patterns are known to the algorithm so there is no new cluster creation. On the other hand, in the rest 600 unseen patterns, the number of clusters scales up to 110 indicating that the ART algorithm is "reinforced" to learn such new patterns.

The circle (o) marked line depicts the C-RLT-2 model, which is trained with *TrainA* and tested with *TestB*. Since the train file is the same as in the C-RLT-1 model, the first generated clusters are the same in number ($|U| = 70$). In the testing phase, we observe a significant difference. ART does not know the second 600 unseen patterns, thus, it is gradually "reinforced" to learn new patterns up to 110 clusters. In the next 600 known patters, C-RLT-2 does not need to learn additional clusters thus it settles at 110 clusters.

We now examine the behavior of the C-RLnT model corresponding to the zero-knowledge training method. The asterisk (*) marked line depicts the training phase (with *TrainB*) followed by the testing phase (with *TestA*) of

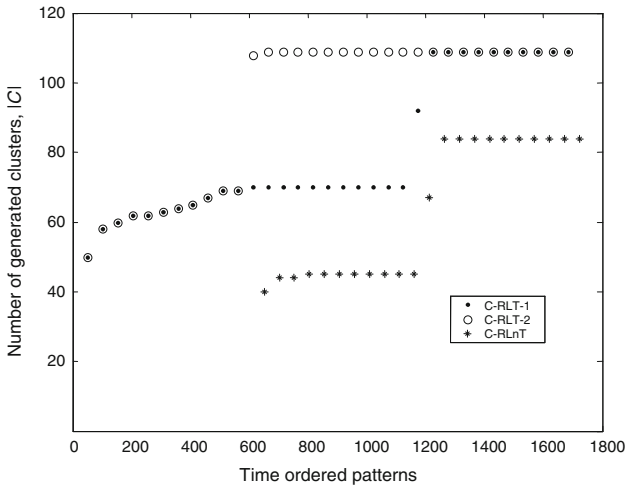


**Fig. 3** Convergence of C-RLT/-RLnT based on the reinforcement learning method

C-RLnT. In this case, we have an incremental ART that does not need to be trained. For technical consistency reasons, it only requires a single pattern, which is the unique cluster in the knowledge base at the beginning. In the testing phase, for the first 600 unseen patterns of *TestA* we observe a progressive cluster creation (up to 45 clusters). For the next 600 unseen patterns, we also observe a gradual cluster creation (up to 85 clusters) followed by convergence. Comparing the C-RLT-1/-2 and C-RLnT models, the latter one achieves the minimum number of clusters (22.72% less storage cost). This is due to the fact that C-RLnT starts learning only from unsuccessful predictions in an incremental way by adapting pre-existing knowledge base to new instances. Nevertheless, we also have to take into account the prediction accuracy in order to reach safe conclusions about the efficiency and effectiveness of the proposed models.

## 5.2 Precision of C-RLT and C-RLnT

In Fig. 4, we examine the precision achieved by the algorithms adopting reinforcement learning. The vertical axis depicts the precision value $p$ achieved during the testing

phase. The point (.) marked line depicts the precision of the C-RLT-1 model trained with *TrainA* and tested with *TestA*. During the test phase, for the first 600 known patterns C-RLT-1 achieves precision value ranging from 97 to 100%. In the next 600 unseen patterns, we observe that for the first instances the precision drops smoothly to 95% and as C-RLT-1 is reinforced to learn, i.e., learn new clusters and optimize the old ones, the precision converges to 96%.

The circle (o) marked line depicts the precision behavior for the C-RLT-2 model tested with *TestB* and trained with *TrainA*. With the first 600 totally unseen patterns during the test phase, C-RLT-2 achieves precision from 26 to 96%. This indicates that the model is still learning during the test phase adopting the reinforcement learning method increasing the precision value. In the next 600 known patterns, the model has nothing to learn and the precision value converges to 96%.

The asterisk (*) marked line depicts the precision behavior of the C-RLnT model tested with *TestA* and trained with *TrainB*. In this case, C-RLnT is trained with only one pattern instance, i.e., the algorithm is fully incremental, thus, all the instances are treated as unseen. In the test phase, for the first 600 patterns, the model achieves precision, which ranges from 25 to 91% (due to the reinforcement mechanism). In the next 600 patterns, we can notice that for the first instances the precision drops smoothly to 88% and as the model is "reinforced" to learn, precision gradually converges to 93%. From the first 200 patterns starting from the 601th pattern, the algorithm updates locally its knowledge structure, as depicted in Fig. 4. Once the algorithm has been trained by such local updates, its precision gradually converges till a new change in the user mobility behavior is detected.

Evidently, the adoption of the training method, i.e., the C-RLT-1 and C-RLT-2 models, yields better precision. However, if we correlate our findings with the results shown in Fig. 3, we infer that a small improvement in precision has an obvious storage cost. Specifically, we need to store 110 clusters, in the case of C-RLT, compared to 85 clusters in the case of C-RLnT (22.72% less storage cost). Furthermore, the user movement patterns can be changed repeatedly over time. Hence, by adopting the training method, one has to regularly train and rebuild the model. Nevertheless, we have to examine the behavior of the C-nRLT and C-nRLnT models in order to decide on the appropriate model for the discussed domain.

### 5.3 Convergence of C-nRLT and C-nRLnT

In Fig. 5, we see the number of clusters that are generated by C-nRLT/-nRLnT assuming the non-reinforcement learning method. The vertical axis indicates the generated clusters during the training and the testing phase. The point (.) marked line depicts the training phase with *TrainA* and the testing phase with *TestA* for the C-nRLT-1 model. As we can see in the training phase, the first 600 patterns (*TrainA*) have generated 70 clusters. In the testing phase, the model knows the first 600 patterns thus there is no new cluster creation. Contrast to the C-RLT-1 model (Section V.B), the remaining 600 unseen patterns cannot be further learned by the C-nRLT-1, thus, the total of clusters remains constant at 70 clusters.

The circle (o) marked line depicts the training phase with *TrainA* and the testing phase with *TestB* for the C-nRLT-2 model. Since the train file is the same (i.e., *TrainA*), the count of the generated clusters is the same (70

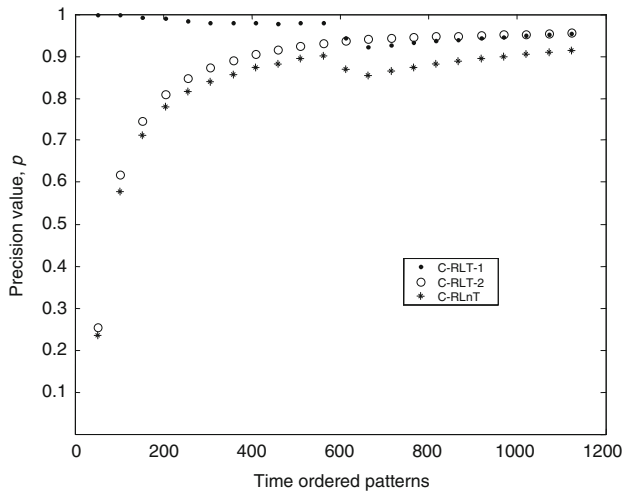


**Fig. 4** Precision of C-RLT/-RLnT based on the reinforcement learning method

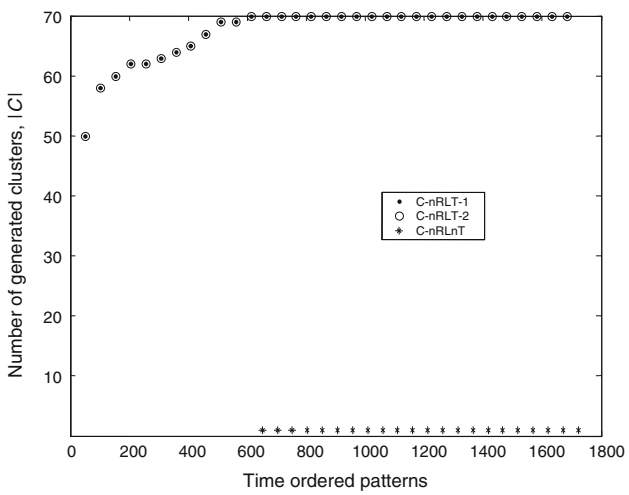


**Fig. 5** Convergence of C-nRLT/-nRLnT based on the non-reinforcement learning method

clusters) as before. In contrast to C-RLT-2, in the testing phase we do not observe a change in the number of clusters. The model does not know the second 600 unseen patterns and is not reinforced to learn new ones, thus, the total cluster count remains constant at 70. In the next 600 known patterns, the model retains the same cluster count (70 clusters).

Finally, the asterisk (*) marked line depicts generated number of clusters for the C-nRLnT model tested with *TestA*. In this case, we have an incremental, non-reinforcement model that is not trained. In the testing phase, for the first 600 unseen patterns we observe that there is no extra cluster generation except the unique cluster (of the *TrainB* file). In addition, for the next 600 unseen patterns, no additional cluster generation is observed. We only notice that the C-nRLT/-nRLnT models achieve a lower amount of clusters than the C-RLT/-RLnT models.

### 5.4 Precision of C-nRLT and C-nRLnT

In Fig. 6, we see the precision of the models adopting the non-reinforcement learning. The vertical axis shows the precision value achieved during the testing phase. The point (.) marked line refers to the C-nRLT-1 model (trained with *TrainA*, tested with *TestA*). In the testing phase, the first 600 known patterns achieve precision levels ranging from 97 to 100%. In the next 600 unseen patterns, we notice that the precision value drops gradually to 59%. This is attributed to the fact that there are no new clusters to optimize the old ones due to the lack of the reinforcement learning mechanism. This indicates that C-nRLT-1 model does not adapt to new knowledge.

The circle (o) marked line refers to the C-nRLT-2 model (trained with *TrainA*, tested with *TestB*). In the testing phase, the first 600 unseen patterns achieve precision levels

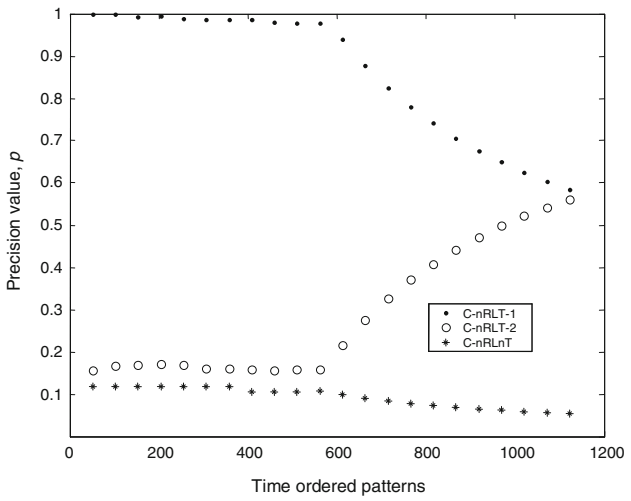


**Fig. 6** Precision of C (-nRLT/-nRLnT) based on the non-reinforcement learning method

**Table 3** Models comparison

| Metric/model | C-RLT | C-RLnT | C-nRLT | C-nRLnT |
|---|---|---|---|---|
| Precision | ++ | + | − | − |
| Storage | − | + | − | ++ |

++ *Excellent performance*, − *Very poor performance*

ranging from 16 to 19% because the reinforcement mechanism is not used and, thus, new clusters cannot derive. In the next 600 known patterns, we notice that the precision value converges to 59%.

Finally, the asterisk (*) marked line depicts the behavior of the precision of the C-nRLnT model (loaded with *TrainB*, tested with *TestA*). In this scenario, the algorithm is fully incremental and all instances are treated as previously unseen. As we can see in the testing phase the first 600 patterns achieve precision levels ranging from 10 to 14% and, in the next 600 pattern instances, we notice that the precision value drops smoothly to 7%. Evidently, the C-nRLT/-nRLnT models achieve much lower precision than their counterparts C-RLT/-RLnT since the former models do not support adaptation.

Therefore, the adoption of the reinforcement learning method for location prediction produces better results than a non-reinforcement algorithm. Each approach has certain advantages and limitations. If the major concern is to keep the storage cost as lightweight as possible, the C-RLnT and C-nRLnT models should be chosen. If the mobile context-aware application aims at maximizing the supported quality of service w.r.t. precision, while keeping the storage cost stable, the C-RLnT model should be adopted. Table 3 summarizes our conclusions with respect to the comparison of the four alternative models discussed throughout the paper. Comparing the four versions of the C algorithm, we can clearly distinguish the C-RLnT model as the most efficient algorithm for location prediction.

## 6 Comparison with Other Models

We compare the C-RLnT model with other known models that can be used for location prediction. Such models implement the Offline *k*Means and Online *k*Means algorithms. Such models require a predefined number of $k > 1$ initial clusters for constructing the corresponding knowledge base. We should stress here that, the greater the $k$ the greater the precision value achieved by Offline and Online *k*Means. In our case, we could set $k = 110$, which is the convergence cluster-count for the C-RL models (Section V). For C-RLnT, we use *TrainB* for the training and *TestA* for the testing phase (such model adopts the zero-knowledge training method). Moreover, for the Offline and

Online *k*Means models we use *TrainA* for the training and *TestA* for the testing phase because both models require $k > 1$ initial clusters.

Figure 7 depicts the precision achieved by the C-RLnT (the point (.) marked line), Offline *k*Means (the asterisk (*) marked line) and Online *k*Means (the circle (o) marked line) models. The horizontal axis represents the ordered instances and the vertical axis represents the achieved precision. We can observe in the first 600 patterns C-RLnT achieves precision levels ranging from 25 to 91% indicating adaptation to new knowledge. This is attributed to the reinforcement mechanism (C-RLnT recognizes and learns new user movements). In the next 600 patterns we notice that for the first instances, the precision drops smoothly to 88% and as the knowledge base adapts to new movements and optimizes the existing ones, precision converges to 93%.

In the case of Offline *k*Means, we observe that for the first 600 patterns, it achieves precision levels ranging from 96 to 98% once the initial clusters are set to $k = 110$. In the next 600 patterns we notice that the precision drops sharply and converges to 57% as the knowledge base is not updated by unseen user movements. By adopting Online *k*Means we observe that for the testing phase (the first 600 patterns) it achieves precision levels ranging from 94 to 97% given the train file *TrainA*. In the next 600 patterns we notice that for the first instances the precision drops rather smoothly to 86% and, as the knowledge base is incrementally adapting to new patterns, the precision value converges to 65%. Evidently, by comparing such three models, the most suitable model for location prediction is the C-RLnT since:

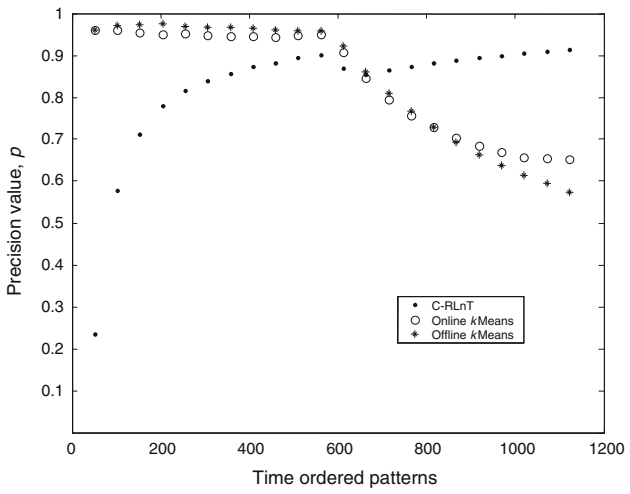


**Fig. 7** Comparison of C-RLnT with the Offline/Online *k*Means models

(i) it achieves greater precision through model adaptation and
(ii) it requires a smaller size of the underlying knowledge base (i.e., less clusters) than the Offline/Online *k*Means models.

Up to this point we have concluded that the C-RLnT model achieves good precision with limited memory requirements, which are very important parameters for mobile context-aware systems. However, we need to perform some tests with C-RLnT in order to determine the best value for the spatiotemporal parameter vigilance $\rho$. In other words, we aim to determine the best values for both spatial $\rho_s$ and temporal $\rho_t$ vigilance coefficients in order to obtain the highest precision with low memory requirements. We introduce the weighted sum $\gamma$ as follows:

$$\gamma = w \cdot p + (1 - w) \cdot (1 - a)$$

where $a$ is the proportion of the generated clusters by the algorithm (i.e., the size of the knowledge base in clusters) out of the total movement patterns (i.e., the size of the database in patterns), that is: $a = |C|/|U|$; $|C|$ is the cardinality of the set $C$. The weight value $w \in [0, 1]$ indicates the importance of precision and memory requirements; a value of $w = 0.5$ assigns equal importance to $a$ and $p$. In our assessment, we set $w = 0.7$. We require that $a$ assumes low values minimizing the storage cost of the algorithm. A low value of $a$ indicates that the applied algorithm appropriately adopts and learns the user movements without retaining redundant information. The value of $\gamma$ indicates which values of $\rho_s$ and $\rho_t$ maximize the precision while, on the same time, minimize the memory requirements. Hence, our aim is to achieve a high value of $\gamma$ indicating an adaptive algorithm with high value of precision along with low storage cost. As illustrated in Fig. 8, we obtain a global maximum value for $\gamma$ once $\rho_s = 100$ m and $\rho_t = 10$ *min* (which are the setting values during the experiments—see Table 3).

## 7 Prior Work

Previous work in the area of mobility prediction includes the model in [7], which uses Naïve Bayes classification over the user movement history. Such model does not deal with fully or semi- random mobility patterns and assumes a normal density distribution for the underlying data. However, such assumptions are not adopted in our model as long as mobility patterns refer to real human traces with unknown distribution. Moreover, the learning automaton in [8] follows a linear reward-penalty reinforcement learning method for location prediction. However, such model does not provide satisfactory prediction accuracy, as reported in

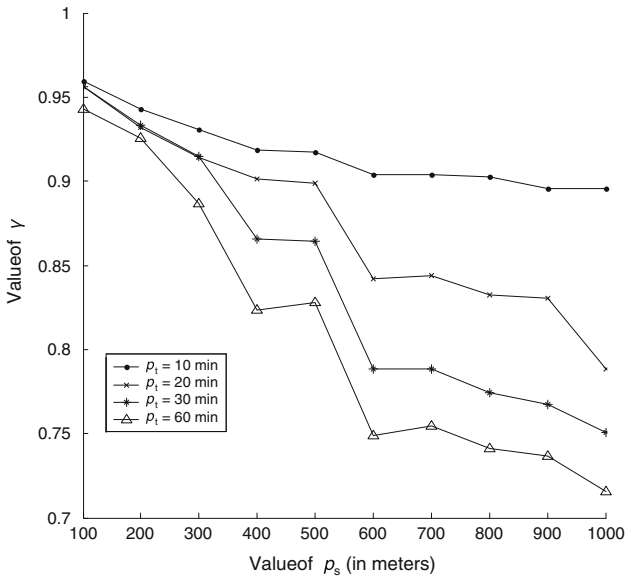


**Fig. 8** The behavior of the $\gamma$ parameter vs. temporal and spatial coefficients of the vigilance threshold

[8]. The authors in [9] apply evidential reasoning in mobility prediction when knowledge on the mobility patterns is not available (i.e., similarly to this paper). However, such model assumes large computational complexity (due to the adopted Dempster-Schafer algorithm) once the count of possible user locations increases and requires detailed user information (e.g., daily profile, preferences, favorite meeting places). In addition, such model is not capable of detecting new patterns and, thus, no updates in the knowledge base. Other methods for predicting trajectory have also been proposed in the literature [10] but these have generally been limited in scope since they consider rectilinear movement patterns only (e.g., highways) and not unknown patterns. A closely related work to ours has been reported in [11], where a GPS system is used to collect location information. The proposed system then automatically clusters GPS data taken into meaningful locations at multiple scales. These locations are then incorporated into a similar Markov model to predict the user's future location. The authors in [16] adopt a data mining approach (i.e., rule extraction) for predicting user locations in mobile environments. This approach achieves prediction accuracy lower than ours (i.e., in the order of 80% for deterministic movement). Moreover, such model dos not provide adaptivity to pattern changes. In [17], the authors adopt a clustering method for the location prediction problem. Prediction accuracy is still low (in the order of 66% for deterministic movements). The authors in [18] introduce a framework where for each user an individual function is computed in order to capture its movement. This approach achieves prediction accuracy lower than ours (i.e., in the order of 70% for deterministic movement). In [19], the authors apply movement rules in mobility prediction given the user's past movement patterns. Prediction accuracy is still low (i.e., in the order of 65% for deterministic movements). The authors in [20] introduce a prediction model that uses grey theory (i.e., a theory used to study uncertainty). This approach achieves prediction accuracy lower than ours (i.e., in the order of 82% for deterministic movement). To the best of our knowledge, there is no prior work reporting an algorithm able to achieve high levels of prediction accuracy and handle pattern changes.

## 8 Conclusions

We presented how ML can be applied to the engineering of mobile context-aware applications for location prediction. Specifically, we proposed an adaptive ML algorithm for location prediction using ART (a special Neural Network Local Model). We introduce two learning methods: one with non-reinforcement learning and one with reinforcement learning. Furthermore, we deal with two training methods for each learning method: in the supervised method the model uses training data in order to make classification and in the zero-knowledge method the model incrementally learns from unsuccessful predictions. We evaluated our models (versions of the proposed algorithm) with different spatial and temporal parameters. We examine the knowledge bases storage cost (i.e., emerged clusters) and the precision measures (prediction accuracy). Our findings indicate that the C-RLnT suits better to context-aware systems. The advantage of C-RLnT is that (1) it does not require pre-existing knowledge in the user movement behavior in order to predict future movements, (2) it adapts its on-line knowledge base to unseen patterns and (3) it does not consumes much memory to store the emerged clusters. For this reason, C-RLnT is quite useful in context-aware applications where no prior knowledge about the user context is available. Furthermore, through experiments, we decide on which vigilance value achieves the appropriate precision w.r.t. memory limitations and prediction error. Finally, in the Neural Networks Local Models literature there are other models (e.g., Self-Organizing Maps) that we have not examined in this paper. We intent to implement and evaluate them with C-RLnT by means of knowledge base requirements, precision of the location prediction and possible adaptation to the built model.

## References


1. A. Dey, Understanding and using context, *Personal and Ubiquitous Computing*, Vol. 5, No. 1, pp. 4–7, 2001.
2. J. Hightower and G. Borriello, Location systems for ubiquitous computing, *IEEE Computer*, Vol. 34, No. 8, pp. 57–66, 2001.
3. E. Alpaydin, *Introduction to Machine Learning*, The MIT Press, Cambridge, 2004.
4. R. Duda, P. Hart and D. Stork, *Pattern Classification*, Wiley-InterscienceNew York, 2001.
5. E. Belogay, C. Cabrelli, U. Molter and R. Shonkwiler, Calculating the Hausdorff distance between curves, *Information Processing Letters*, Vol. 64, No. 1, pp. 17–22, 1997.
6. Site: http://www.openstreetmap.org/traces/tag/Denmark.
7. S. Choi, K. G. Shin, *Predictive and adaptive bandwidth reservation for hand-offs in QoS-sensitive cellular networks*, *ACM SIGCOMM*, 1998.
8. S. Hadjiefthymiades, L. Merakos, Proxies+path prediction: improving web service provision in wireless-mobile communications, *ACM/Kluwer Mobile Networks and Applications, Special Issue on Mobile and Wireless Data Management*, Vol. 8, No. 4, pp. 389–399, 2003.
9. A. Karmouch, N. Samaan and A. Mobility, Prediction architecture based on contextual knowledge and spatial conceptual maps, *IEEE Transactions on Mobile Computing*, Vol. 4, No. 6, pp. 537–551, 2005.
10. R. Viayan and J. Holtman, A model for analyzing handoff algorithms, *IEEE Transactions on Vehicular Technology*, Vol. 42, No. 3, pp. 351–356, 1993.
11. D. Ashbrook and T.Starner, Learning Significant Locations and Predicting User Movement with GPS, *Proc. Sixth Int'l Symp. Wearable Computes (ISWC 2002)*, pp. 101–108, Oct. 2002.
12. I. Priggouris, E. Zervas and S. Hadjiefthymiades, Location based network resource management. In Ismail Khalil Ibrahim, editor. *Handbook of Research on Mobile Multimedia*, Idea Group IncHershey, 2006.
13. K.M.Curewitz, P. Krishnan, and J.S.Vitter, Practical Prefetching via Data Compression, In *Proceedings of ACM SIGMOD*, pp. 257–266, 1993.
14. K. Narendra and M. A. L. Thathachar, *Learning Automata–An Introduction*, Prentice HallEnglewood Cliffs, 1989.
15. R. Jain Cheng and E. van den Berg, *Location prediction algorithms for mobile wireless systems, Wireless Internet handbook: technologies, standards, and application*, CRC PressBoca Raton, 2003. pp. 245–263.
16. G. Yavas, D. Katsaros, O. Ulusoy and Y. Manolopoulos, A data mining approach for location prediction in mobile environments, *Data and Knowledge Engineering*, Vol. 54, No. 2, pp. 121–146, 2005.
17. D. Katsaros, A. Nanopoulos, M. Karakaya, G. Yavas, O. Ulusoy and Y. Manolopoulos, Clustering mobile trajectories for resource allocation in mobile environments, In *Proceedings IDA*, pp. 319–329, 2003.
18. Y. Tao, C. Faloutsos, D. Papadias and B. Liu, *Prediction and Indexing of Moving Objects with Unknown Motion Patterns, ACM SIGMOD*, 2004.
19. V.T.H. Nhan and K.H. Ryu, *Future Location Prediction of Moving Objects Based on Movement Rules*, Springer ICIC, LNCIS 344, pp. 875–881, 2006.
20. Y. Xiao, H. Zhang, H. Wang, Location Prediction for Tracking Moving Objects Based on Grey Theory, *IEEE FSKD*, 2007.


## Author Biographies

**Theodoros Anagnostopoulos** received his B.Sc. in Informatics from the Department of Informatics at the Technical Educational Institution of Athens, Athens, Greece in 1997. He also graduated the Department of Informatics at the Athens University of Economics and Business (AUEB), Athens, Greece in 2001. He was awarded a M.Sc. in Information Systems from the same Department in 2002. Currently, he is a Ph.D. candidate in the National and Kapodistrian University of Athens (NKUA), Department of Informatics and Telecommunications. His research interests are in the areas of Pervasive Computing and Machine Learning. He is a member of the Pervasive Computing Research Group of NKUA.

**Christos Anagnostopoulos** has received his B.Sc. in Computer Science from the Department of Informatics and Telecommunications at the National and Kapodistrian University of Athens (NKUA), Greece, in 2001 and his M.Sc. in Computer Science, Advanced Information Systems from the same department in 2003. He holds a Ph.D. in Modelling Mobile and Distributed Computing Systems (2008) from NKUA. His research interest is focused on Mobile and Distributed Computing Systems, Context- aware Computing, Semantic Web and Ontological Engineering. He had also participated in projects realized in the context of European Union Programs.

**Stathes Hadjiefthymiades** received his B.Sc., M.Sc. and Ph.D. in Informatics and Telecommunications from the Department of Informatics & Telecommunications (DIT) of the University of Athens (UoA), Athens, Greece. He also received a joint engineering-economics M.Sc. degree from the National Technical University of Athens. In 1992 he joined the Greek consulting firm Advanced Services Group, Ltd., as an analyst/developer of telematic applications and systems. In 1995 he became a member of the Communication Networks Laboratory of UoA. During the period 2001–2002, he served as a visiting assistant professor at the University of Aegean, Department of Information and Communication

Systems Engineering. In 2002 he joined the faculty of the Hellenic Open University (Department of Informatics), Patras, Greece, as an assistant professor. Since the beginning of 2004, he belongs to the faculty of UoA, DIT where he presently is an assistant professor. He has participated in numerous projects realized in the context of EU programmes and national initiatives. His research interests are in the areas of mobile, pervasive computing, web systems engineering, and networked multimedia applications. He is the author of over 150 publications in these areas.