Generation of human computational models with machine learning

Francisco Campuzano [a,*], Teresa Garcia-Valverde [a], Juan A. Botia [a], Emilio Serrano [b]

[a] Universidad de Murcia, Facultad de Informática, Campus Universitario de Espinardo, 30100 Murcia, Spain
[b] Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros de Telecomunicación, Ciudad Universitaria, 28040 Madrid, Spain

A B S T R A C T

Services in smart environments pursue to increase the quality of people's lives. The most important issues when developing this kind of environments is testing and validating such services. These tasks usually imply high costs and annoying or unfeasible real-world test-ing. In such cases, artificial societies may be used to simulate the smart environment (i.e. physical environment, equipment and humans). With this aim, the CHROMUBE methodol-ogy guides test engineers when modeling human beings. Such models reproduce behaviors which are highly similar to the real ones. Originally, these models are based on automata whose transitions are governed by random variables. Automaton's structure and the prob-ability distribution functions of each random variable are determined by a manual test and error process. In this paper, it is presented an alternative extension of this methodology which avoids the said manual process. It is based on learning human behavior patterns automatically from sensor data by using machine learning techniques. The presented approach has been tested on a real scenario, where this extension has given highly accurate human behavior models.

1. Introduction

A smart environment [13] provides a set of software and hardware elements that support an intelligent and unobtrusive interaction between environment and users. These intelligent environments acquire context knowledge from the physical world, i.e. they are Context-Aware, and provide services that support daily user tasks in an adaptive way. The main goal of these services is to augment quality of people's lives. Before their deployment, services need to be tested and validated. Testing [35] is the process by which a program is executed with the intent of finding errors in the code, whereas validation [15] is the process of evaluating a system or component during or at the end of the development process in order to determine whether it satisfies specific requirements. A common approach to test and validate smart environments is living labs. A living lab [42,37] is a physical environment where a system under test (SUT) is deployed, tested and validated. The living lab environment must emulate the real environment where the SUT is going to be deployed, including users and their interactions with the SUT. But living labs present an important lack: sometimes they are not feasible, e.g. when the environment to be emulated includes a large number of users. Moreover, a physical deployment may involve very high costs.

* Corresponding author. Tel.: +34 868887317.
E-mail addresses: fjcampuzano@um.es (F. Campuzano), mtgarcia@um.es (T. Garcia-Valverde), juanbot@um.es (J.A. Botia), eserrano@gsi.dit.upm.es (E. Serrano).

As an alternative, in validation scenarios where the living lab option is not applicable, new approaches based on social simulation are being used. Social simulation [18] implies the imitation of real social phenomena, human or not, over time by using simulation models on a computer. A virtual model of a smart environment scenario may be created by including simulated users, sensors and appliances, whereas some components of the model may be real (i.e. the SUT or the middleware software). The use of simulated users is an interesting alternative as long as the generated human models are realistic enough. For this purpose, the CHROMUBE methodology (CHROnobiology for Modeling hUman BEhavior) was proposed in a previous work [9]. This methodology describes how a developer can analyze real data on a target subject (TS) by means of some chronobiological techniques. The results of this analysis are employed to create Computational Models of Human Behavior (CMHB) that imitate the behavior of the TS. The underlying CMHB model is an automaton whose states represent behaviors. Every behavior (i.e. automaton state) requires some input parameters the developer must configure, e.g. type of behavior, priority or hour intervals. The most important parameter of every behavior is a probability distribution function (pdf) that generates the time instants of the day when the CMHB is going to manifest such behavior. Considering these time instants and other parameters as the priority, an interpreter is in charge of managing the transitions between states throughout the day. The developer can obtain some configuration parameters from the data and the chronobiological representations (e.g. plexograms) and they must be adjusted to coordinate the different behaviors of the CMHB properly. This process may be difficult, as it requires some expert knowledge and trial and error tests until a model becomes realistic enough to fit to the developer's purpose.

This paper proposes an alternative extension of CHROMUBE which tries to avoid the initial analysis of real data and the manual configuration of the model by means of a machine learning process. The main hypothesis behind this extension is the following. A machine learning technique (e.g. a neural network) can learn behavior patterns directly from behavior data and imitate the TS. If this hypothesis is validated, it can represent the first step for designing an automatic process in which the data analysis performed by the developer is no longer necessary and the models are automatically generated and validated. These features would produce an important improvement to the generation of realistic CMHBs for simulation. Numerous simulation tools (e.g. Ubiwise [5], UbiREAL [38], TATUS [39], UbikSim [44]) could improve their performance by including an automatic generator of human models.

The realistic human models generated in this paper are intended to be used in simulated scenarios with different purposes. One of them is the validation of smart environments [10], e.g. CMHBs previously produced by CHROMUBE were used prior to the pilot experience of the Necesity system.[1] Necesity is an Ambient Assisted Living (AAL) system which deploys a set of sensors (i.e. motion sensors, presence sensors and open-door detectors) in all the rooms of a house where an elder person lives. Events generated by sensors are registered in a log. As a consequence, the system is able to know the elder's activity and location at any time. One of the most important elements of Necesity is the adaptation algorithm used to learn from the user's habits and to detect when the user is suffering an unexpected situation, e.g. a fall. The CMHBs were useful for testing the generalization capacity of the algorithm in a simulated environment. Thanks to this, the duration of the pilot experience was considerably reduced. Nowadays CHROMUBE is being implemented on other application fields; CMHBs are intended to be used in simulations of several scenarios such as emergency evacuations in geriatrics or hospitals.

The present paper is structured as follows. Section revises some related works Section 2. Section 3 outlines the basic ideas of CHROMUBE and presents the machine learning based alternative. Section 4 applies the different steps of CHROMUBE machine learning process (i.e. data adaptation, machine learning process, generation and validation) to an AAL smart environment. Section 5 presents conclusions and future works.

2. Related work

As a result of the massive presence of sensors in everyday life deployments, more and more applications and services are concerned with the user's behavior. All these environments may benefit from the use of CHROMUBE. Some examples of environments with heterogeneous sensors are cameras [22,16], wearable sensors [20], unobstrusive sensors (i.e. presence, open door, pressure sensors) [14,25], sensors integrated in mobile phones or PDAs [29,40], etc. Some scenarios in which these sensors are used to monitor the user and from which behavior patterns can be learned are intelligent environments [8], human robot interaction [6] or Ambient Assisted Living (AAL) [10].

In this paper, an alternative extension of the CHROMUBE methodology is proposed. Such extension offers an alternative manner to create CMHBs that imitate a TS. The alternative proposes using machine learning to generate a CMHB. Machine learning has been extensively used to create behavior models from sensor data [36,27]. Examples of machine learning techniques used include Partially Observable Markov Models [21], Kernel density estimators [27], Naive Bayes [17], boosting ensembles like AdaBoost [28], inductive logic programming [26], decision tree classifiers [4] or Hidden Markov Models [25,11,24].

In this paper, the application of the new CHROMUBE extension is exemplified by using ANNs as the machine learning technique. ANNs are a common method used to interpret data obtained from sensors. An example is the work of Atkeson et al. [2], where data from both human and robot arm movements are collected and neural networks are used to correlate inputs (i.e. human movements) and outputs (i.e. robot movements) in order to create robots that move like humans. Another

[1] http://www.necesity.net/, last access: October 2012.

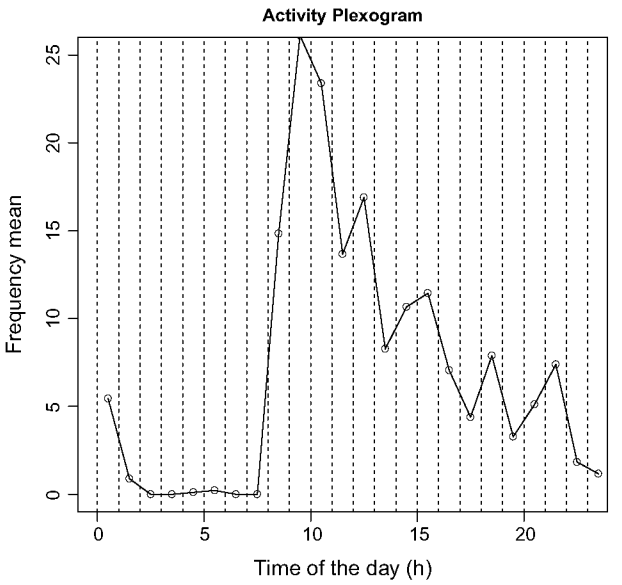


Fig. 1. A plexogram that represents the activity rate of a person. This activity rate is represented by the number of times, in average, the person moves from one room to another at home.

example is the work of Al Mashhadany [1]. It uses ANNs to generate a virtual human, but considering only simple postures, not behaviors, in this case. There are also some proposals which apply ANNs to location data. Azam et al. [3] present a prediction system whose goal is to detect whether an elder person may be having any kind of trouble depending on his/her location. The inputs for the ANN are location, hour of the day and day of the week. The output is a behavioral level between 1 and 0 in which 1 represents a normal situation and 0 represents abnormality. Alternative works go in the direction of using ANNs for modeling the user to predict location [45,46]. This paper goes beyond as it applies ANNs to sensor data. CHROMUBE uses the resulting trained ANN to generate a CMHB that behaves in a similar way to the TS. This approach is a complete process for the automatic generation of faithful computational models of human behaviors in order to get realistic simulations.

3. Proposal of the CHROMUBE's extension

CHROMUBE addresses the problem of obtaining computational human models which are useful for realistic simulations in general and for service testing and validation in particular. This methodology is capable of generating and validating a realistic CMHB, i.e. one that reproduces a behavior highly similar to that used as a source. The approach is based on tools from two rather different but somewhat related disciplines.

On the one hand, statistical techniques allow for the construction of the model with a probabilistic automaton. They are used to assess the validity of the pdf for each transition between states. This validation is based on source data resulting from sensors deployed at the smart environment. This source data reflect the real behavior of the TS.

On the other hand, behavior representation tools from Chronobiology are used to characterize the subject's behavior. Chronobiology [19] is the field of biology that studies the mechanisms and alterations of each organism's temporal structure. Plexograms are the behavior representation tool used in CHROMUBE. A plexogram characterizes the activity rate of humans or animals and how it varies depending on time. Activity rate is represented by a curve consisting of a number of points. Each point corresponds to an interval of the day. The day is divided into n intervals, and for each interval, a mean is obtained with all the sample's activity values within the same interval for all the days. Thus, an increase in the curve represents an increase of the subject's activity in the corresponding period, and it occurs analogously when it decreases.

An example of a plexogram is shown in Fig. 1. In this case, the curve consists of 24 points. Every point represents an hour of the day, although more precise intervals could be defined. The TS of the figure represents a person who probably sleeps during the night (from 1 a.m. to 7 a.m. approximately) and has his maximum activity rate around 9 a.m.

Plexograms are interesting because they provide a means to display visual representations of the TS's behavior. As a consequence, plexograms are used as the basic means for the description of behavior. Thus, the ultimate goal of CHROMUBE is to produce CMHBs which reproduce plexograms which are highly similar to those of the TS. In the original CHROMUBE proposal [9], the responsibility of tuning the CMHB so as to get the required plexograms falls on the developer. But as it has been pointed out above, this must be done by means of a tedious test and error process. Thus, CHROMUBE has been extended in order to provide automated means to generate CHMBs based on machine learning. Fig. 2 shows a diagram representing the overall methodology including the proposed extension. Such extension includes steps 3b, 4b and 5b. The rest of the steps remains unchanged.

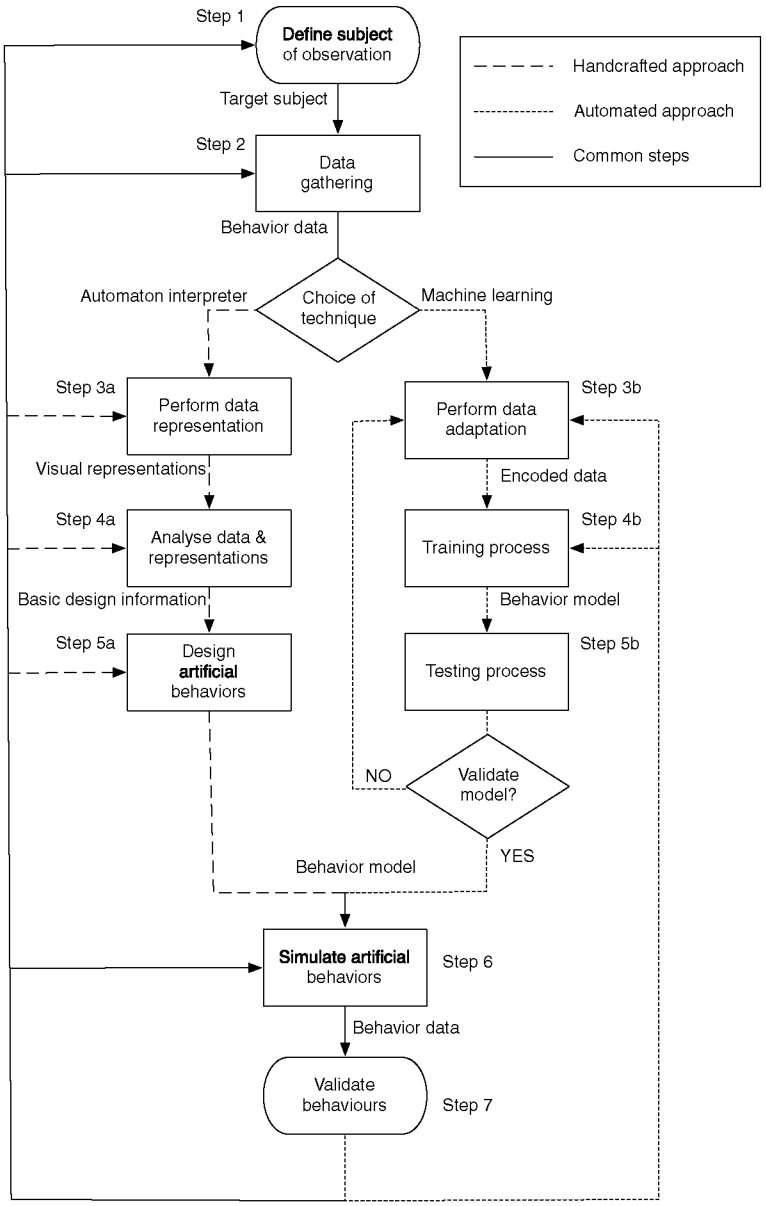


Fig. 2. CHROMUBE steps to obtain computational models of human behavior.

In the next paragraphs, the steps to be followed in this methodology are revised. It is worth noting that the methodology has two branches (i.e. steps 3, 4 and 5, `a` and `b`). The one at the left side represents the generation of CMHBs by means of a manual probabilistic automaton. The one at the right side includes the proposal presented in this paper: generation of CMHBs by means of machine learning techniques. Step 1 consists in defining the subject of observation (i.e. the TS). In this step, the human to be modeled must be selected. It must include a human whose presence and action influence the behavior of the smart environment. Step 2 corresponds to data gathering from sensors deployed at the smart environment. Such sensors are the *eyes* through which the system perceives the human and sensor's data granularity settles down the abstraction level of the CMHB to be produced (i.e. the CMHB is not required to do anything no sensor will perceive).[2]

[2] In this paper, it is assumed that sensors are the only data source. But there are cases in which CMHBs are used to study how to deploy a smart environment instead of testing it once it is deployed. In these cases, no sensor data are available and it is recommended instead to extract information from other sources such as experts who know how the TS's behave daily (e.g. the manager of a geriatric about its residents). We are currently working on this particular case regarding step 2.

Depending on the kind of data obtained and the developer's validation purposes, one of the two proposed approaches must be chosen (choice of technique). If the manual probabilistic automaton option is selected, steps 3a, 4a and 5a are required to produce a CMHB. Step 3a generates a visual representation from the data gathered in step 2. These visual representations should be performed by means of chronobiological techniques as plexograms or actograms.[3] These techniques are the basis for step 4a, where a detailed analysis of both data and representations is needed to extract behavior patterns. Statistical approaches allow for finding probability distribution functions (pdfs) governing transitions at the automaton. Then, a basic design description including pdfs and their associated parameters (e.g. a λ value for the exponential pdf) for every detected behavior pattern is generated. Such description is used in step 5a, where a suitable CMHB that reproduces the behavior of the TS is generated. More details about this part can be found in [9].

In case of choosing the machine learning based branch of CHROMUBE to produce the CMHB, steps 3b, 4b and 5b must be followed. Step 3b consists in obtaining the most relevant information from behavior data. These data should be encoded afterwards to be transformed into an appropriate input for the specific supervised learning technique (e.g. neural networks, decision trees, etc.). These techniques are characterized by the usage of a training data set composed of training examples. Every training example consists of some input variables and one or more output variables. Step 4b entails using the encoded data to obtain a reliable behavior model which is tested in step 5b. The testing process assures the correctness of the behavior model created. If the results are not favorable, it means the behavior model is not suitable enough to represent a CMHB. Thus, the developer may repeat the training process including some changes in it. Changes can be introduced in the data encoding step, or they can consist in choosing a different machine learning technique or maintaining the learning technique but modifying some parameters of the training process. This iterative process may be repeated until quality of the behavior model proves to be enough in step 5b.

Steps 6 and 7 are also common to both approaches. Step 6 simulates the behavior model generated by either step 5a or 5b. The step receives a behavior model and returns behavior data in the same format than the one generated in step 2. Notice that each behavior model has a concrete means for simulation. For example, the probabilistic automaton has an interpreter capable of generating a sequence of event changes resembling the TS's behavior [9]. See Section 4.3 for details about the way the CMHB produced by means of a machine learning technique (i.e. a regression model) can generate behavior data. Finally, a validation of the CMHB is performed in step 7. It is important to obtain accurate representations of the CMHBs when comparing them with the representations of the original TS. In this step, the use of chronobiological representations of the CMHBs and the TS's behavior can help the developer to validate the CMHBs and to select the CMHB whose behavior is most similar to that of the TS. The next section shows the application of the machine learning based branch of CHROMUBE to illustrate its applicability. A previous work [9] shows the proposal and applicability of the probabilistic automaton based CHROMUBE branch to the same real problem.

4. Applying methodology in the ambient assisted living domain

In this section, an example of the application of CHROMUBE is shown. It exemplifies the application of the machine learning based branch of CHROMUBE to the AAL domain which was presented in the last part of the introduction. Step 1 involves selecting the subject of observation. As it was mentioned before, the TS is an elder living alone and independently at his/her own house. In this work, two TSs of the same kind are used to illustrate the use of CHROMUBE. The houses where the two TSs live have five rooms each (living room, bathroom, bedroom, kitchen and corridor). Step 2 consists in data gathering. Data are generated at an AAL system called Necesity. Necesity is composed of a wireless sensor network deployed at the houses where the two selected TSs live and a minicomputer with wireless connection that serves as a base station. The sensors generate information about the TS's activity, which is measured in terms of room changes. Once sensor data are processed at the minicomputer, an historical log is generated. Its data represent an elder person living independently at home during 20 days. The logs obtained from the smart environment are the base for the generation of the CMHBs. Every log entry is an activity mark. Such marks are then transformed into training examples for the learning algorithm, and this learning algorithm will produce a model of the TS's behavior to be used as the basis for the CMHB. Such machine learning model works in the following way. Let us suppose that the simulated TS is, in a concrete instant of time t, at the living room. The model is used to predict where the simulated TS will go to when leaving the living room. But it will also predict how much time the person is going to spend at the living room before moving to the next one. Thus, the output of step 2 is the behavior data needed to generate such machine learning model.

Before describing steps 3b, 4b and 5b, some details about the generic definition of the learning process are given. The TS's daily behavior to be reproduced can be expressed as a function $f_{TS}(loc, t) \rightarrow (loc', ts)$ where loc is the location where the TS is at the moment t of the day and loc' is the next location the TS goes to by the time elapsed in ts. The learning problem may be defined in terms of searching for a target function, precisely f_{TS} as it perfectly encodes the TS's behavior. Given that the real f_{TS} cannot be reached, the task is redefined to search for the $\hat{f}_{TS}$ that minimizes a generic error function, $\Delta(f_{TS}, \hat{f}_{TS})$. Thus, in

[3] Actograms are chronobiological representations where each daily activity pattern is superimposed to the previous daily pattern so as to compound the whole activity lectures of the subject. The patterns are composed of marks that represent activity and blank spaces that stand for inactivity. Actograms could serve as an alternative to plexograms as they show more detailed activity patterns. However, in this work activity patterns are unnecessary as they do not need to be interpreted by the developer.

order to define the learning process totally, the form of $\hat{f}_{TS}$ (e.g. a neural network, a decision tree, etc.), the search process used to find a good $\hat{f}_{TS}$ and the definition for Δ function are needed.

The form of $\hat{f}_{TS}$ selected for this example is an Artificial Neural Network (ANN) [7]. For certain types of problems, such as learning to model complex real-world from sensor data, ANNs are among the most effective learning techniques currently known. Due to the nature of the CHROMUBE extension presented in this paper, any supervised learning technique could be used. Other alternatives to this technique are Bayesian Learning, Fuzzy Learning or Decision Trees [33]. Precisely, these machine learning techniques can generate interpretable models, however they are not required in this particular application domain. Instead, the main goal of the CMHB is to behave as similar as possible to the TS during a simulation. No understanding of the internal dynamics of the CMHB is required. Moreover, ANNs, as they are used here, allow for an automatic training, evaluation and model selection process to be integrated within CHROMUBE. Thus, the choice is clear.

In a previous work [9], the form of $\hat{f}_{TS}$ was an automaton whose states represented human behaviors. In this example, $\hat{f}_{TS} \in F$ where F is a set that contains all the possible ANNs which present the same configuration of input and output nodes. The following subsections explain how behavior data are used to learn the TS's behaviors and therefore, to emulate them once the ANN is properly trained. In other words, it consists in minimizing Δ by obtaining an ANN (i.e. $\hat{f}_{TS}$) as similar as possible to the TS behavior (i.e. f_{TS}). The error function Δ is defined in Section 4.4.

4.1. Adaptation of behavior data

The training data set can be denoted as the set $\{(\vec{x}, \vec{y})\}$ where $\vec{x}$ refers to the input variables (i.e. the covariates) and $\vec{y}$ refers to the set of output variables (i.e. the response variables). The basic measure used in this study to reflect the TS activity level is based on location changes (i.e. if the subject moves from one room to another, a record of activity should be registered). Given such abstraction level for behavior description, it is obvious that the same abstraction level should be deployed by the CMHBs. Thus, in this case the CMHB is a model of changes between two different rooms. In a previous work involving the same TSs [9], sensor data from Necesity were obtained in the form of log entries. In this work, the same scenario is addressed. Thus, the same data are used in order to compare the two different approaches of CHROMUBE methodology, the original one and the machine learning based alternative.

Fig. 3 shows an extract of the behavior data. Every log entry provided by Necesity is a pair (loc, t), where loc is the location towards which the TS moves at time t. The format of t is the number of seconds elapsed since 0:00 h. This data representation contains only log entries which indicate when the subject moves from one room to another. Each pair contains the new location the subject goes to and its associated timestamp. Thus, every one of these log entries corresponds to a room change. Let $L = \{(loc_1, t_1), (loc_2, t_2), \ldots, (loc_n, t_n)\}$ be the sequence of log entries in which loc_i represents the location where the subject moves at time t_i. L is the training data from which the ANN is going to learn the elder's activity rate during the day.

A peculiarity of the training data set arises when a TS shows long inactivity periods. It is necessary to create new log entries during these inactivity periods in order to train the ANN in a proper way. The ANN should learn the TS's routines for the whole day period. And to be able of doing this, ANNs need training data for the whole 24 h period. Instead, what Necesity generates is a set of sensor events spared throughout the day. Thus, if there are long periods (e.g. more than five minutes) with little or no training data at all, the ANN will not learn properly how the TS behaves during that period. To solve this, new artificial tuples were included every τ seconds into the training data. This condition of regular intervals must be accomplished. Thus a new log history including L and some new tuples must be generated. These new tuples include a timestamp which is a multiple of τ and their location is the same as the previous one. Let τ be a predefined time value measured in seconds which is useful to split a day timeline in $86400/\tau$ regular intervals, being 86,400 the total number of seconds on a 24 h period. Then, let $L' = \{(loc'_1, t'_1), (loc'_2, t'_2), \ldots, (loc'_m, t'_m)\}$ be a set of tuples where $t'_i - t'_{i-1} \leqslant \tau$ and $L \subset L'$. The value of τ affects the size of the training data set. For that reason, the developer should define a value of τ that implies a compromise between the amount of training data and its granularity. In this work, τ was set to 120 s. Let us see an example based on how to generate L' from the example of L shown in Fig. 3. Fig. 4 shows the resulting log L' which includes some new log entries at the interval between timestamps 1397 and 28,899. In consequence, the ANN has enough data to learn that the TS should be at the bedroom during this time interval.

4.1.1. Input encoding

Given that the ANN input must be adapted from the form of L' to the typical format needed by the inputs of the network, a key issue is how to encode this log into effective training data. On the one hand, for a TS whose house contains r different

```
livingroom,78815
bathroom,86334
livingroom,643
corridor,1123
bedroom,1397
corridor,28899
livingroom,29259
```

Fig. 3. An extract of the log L provided by Necesity. It consists of pairs (location, timestamp).

```
livingroom,643
corridor,1123
corridor,1200
bedroom,1397
bedroom,2400
bedroom,3600
...
bedroom,28800
corridor,28899
livingroom,29259
```

Fig. 4. An extract of log L', including the originals pairs from L and new pairs multiple of τ.

rooms, the location input is represented as a binary vector with the form $loc' = (l_1, l_2, \ldots l_r)$, where r is the number of locations (i.e. rooms) in the house and

$$l_k = \begin{cases} 1, & \text{if TS is at location } k \\ 0, & \text{otherwise.} \end{cases}$$

On the other hand, the original timestamp values ranging from 1 to 86,400 are linearly scaled to $(0, 1)$. Let t' be the original timestamp measured in seconds, then $t'' = t'/86,400$ where t'' is the encoded timestamp and 86,400 is the maximum possible value of timestamp.

Thus, let (loc', t') be the original tuple from L' and let $\vec{x} = (l_1, l_2, \ldots l_r, t'')$ be its corresponding encoded tuple. $\vec{x}$ provides the ANN with enough information in the appropriate format to learn the TS's context (i.e. the location where the subject is at every time of day).

4.1.2. Output encoding

Defining training data outputs is a more complex task. An ANN always computes the same output for a given input. It is a deterministic model. However, human behavior is not deterministic. Such non deterministic behavior should also be reflected by the ANN by means of some probabilistic mechanism. Then, the ANN should generate an output which contains a probabilistic value for every one of the r locations at the TSs' house. Let $\vec{y} = (p_1, p_2, \ldots, p_r, ts)$ be an output produced by the ANN, where ts is the amount of time the person will take to move to the next room, and every p_k represents the probability for the TS to move to location k. In this case, a softmax function must be applied to values of p_k in order to ensure the condition $\sum_{k=1}^{r} p_k = 1$ being r the number of rooms. The softmax function returns a new tuple $\vec{y'} = (p'_1, p'_2, \ldots, p'_r, ts)$ where $p'_k = \frac{p_k}{\sum_{i=1}^{r} p_i}$.

The pseudocode at Fig. 5 shows how to calculate p_k and ts from the L' log. Let us suppose we want to encode the output of a tuple $(loc'_i, t'_i) \in L'$. Then, we iterate through all the days of log L' (line 14) to detect all the existing tuples $(loc'_j, t'_j) \in L'$ where the TS is at the same location $(loc'_i = loc'_j)$ within the same time interval $t'_j \in (t'_i - \delta, t'_i + \delta)$, all the days (line 15). For each one of these tuples, a new tuple with the form $(nextLocation, nextTime)$ is created in order to represent the transition between (loc'_j, t'_j) and the following tuple of $L', (loc'_{j+1}, t'_{j+1})$. The value of *nextLocation* is equal to the location the TS moves to $(nextLocation = loc'_{j+1})$ (line 16) and the value of *nextTime* is the time difference between t'_j and t'_{j+1} (line 17). Every tuple $(nextLocation, nextTime)$ is added to a set $Transitions_{(loc'_i, t'_i)}$ (line 18). In summary, $Transitions_{(loc'_i, t'_i)}$ contains an entry for each location the TS moves to at a concrete hour interval from location loc'_i during the study and how much time the TS spends at loc'_i before moving to each one of the next locations. Every value of p_k is an estimate of the probability of going to room k. The number of occurrences of k in $Transitions_{(loc'_i, t'_i)}$ is compared with the total size of the set (line 22) and p_k is generated.

Once $p'_1, p'_2, \ldots, p'_r$ are generated, the ts value (i.e. the time the TS will take to go to the next room) must be estimated. This is simply done by using the arithmetic mean of all the ts values of $Transitions_{(loc'_i, t'_i)}$ (line 26).

4.1.3. Applicability of input and output encoding

It has been explained how to adapt raw sensor data so as to be used in the required machine learning experiments. At this point, it is worth noting that a neural network model is used for learning from data. However, the encoding process explained above is the same for any regression model capable of taking values in R^n and generating values in R^m (e.g. any other universal approximation method like, for example, a fuzzy rule set model or a multivariate regression tree) being R the domain of real numbers. Thus, input and output encoding used in this paper is always applicable regardless of the machine learning technique used. On the other hand, Section 4.2 introduces how to train and evaluate the model. Thus, most of the details included there are attached to the use of neural networks and they must not be generalized when using other techniques.

```
1  Let L' be a set of behavior data tuples with the form
     (loc', t').
2  Let (loc'_i, t'_i) be the tuple of L' to encode.
3  Let Transitions_(loc'_i,t'_i) be a set of transitions data tuples
     with the form (loc', ts), where ts is a time difference
     between two values of t'.
4  Let δ be a time constant with a small value.
5  Let r be the number of locations in the house.
6  Let k be an array with all possible locations,
     k = {k_1, k_2, ..., k_r}.
7  Let T be a predefined threshold.
8  Let count(S, l) be a function that returns the number of
     tuples in the set S whose location is equal to l.
9  Let getTS(S) be a function that return an array with all
     the ts values of the set S.
10 Let mean(a) be a function that returns the arithmetic
     mean of all the values of a.
11
12 Transitions_(loc'_i,t'_i) ← ∅
13 //Probabilities encoding
14 for every (loc'_j, t'_j) ∈ L'
15   if loc'_j == loc'_i and t'_j ∈ [t'_i − δ, t'_i + δ] then
16     nextLocation ← loc'_{j+1}
17     nextTime ← t'_{j+1} − t'_j
18     Transitions_(loc'_i,t'_i) ← Transitions_(loc'_i,t'_i) + (nextLocation, nextTime)
19   endif
20 endfor
21 for k ← 1..r
22   p_k = count(Transitions_(loc'_i,t'_i), k) / size(Transitions_(loc'_i,t'_i))
23 end
24 //Timestamp encoding
25 timestamps ← getTS(Transitions_(loc'_i,t'_i))
26 ts ← mean(timestamps)
```

Fig. 5. Pseudocode of the output encoding for each tuple.

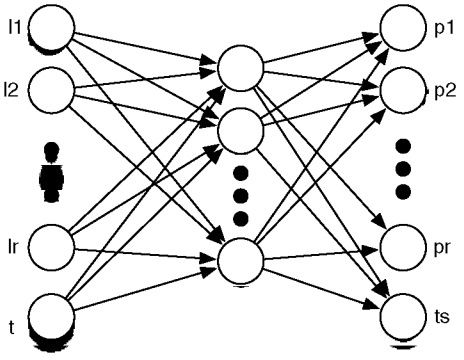


Fig. 6. The ANN structure used to generate CMHBs.

4.2. Training and evaluating the behavior model

This section shows how to configure the ANN learning process to create a realistic CMHB. But prior to this, and given the input and output encoding just commented, Fig. 6 shows the ANN structure. Please note that it is a feed forward ANN (every node is a sigmoidal node connected with all the nodes of the next layer). The ANN architecture is a multilayer perceptron with a single hidden layer. This is a conventional ANN architecture with inputs at the leftmost layer and output at the rightmost layer. All arcs connecting two nodes i,j have a weight w_{ij} indicating the relevance of such connection. The $\hat{f}_{TS}$ is this network and the most important parameters of $\hat{f}_{TS}$ are the set of w_{ij}. The learning task is oriented towards obtaining a good $\{w_{ij}\}$ set that minimizes $\Delta(f_{TS},\hat{f}_{TS})$, the error function.

In the first place, an ANN training algorithm had to be chosen. The selected learning algorithm was an improved version of Backpropagation [31,41]. The Backpropagation algorithm has been proven successfully in many practical problems such as

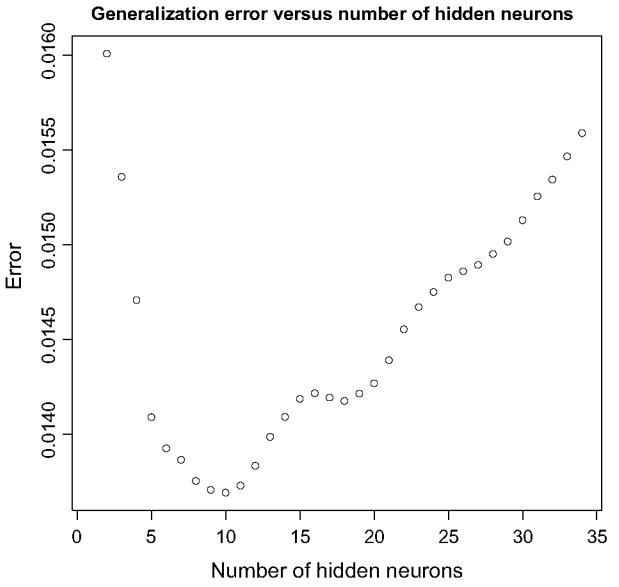


Fig. 7. Best generalization error obtained from ANNs with a varying number of hidden neurons.

learning to recognize handwritten characters and learning robot control strategies. This algorithm uses gradient descent to tune network parameters to best fit a training set of input–output pairs.

The set of w_{ij} values is initialized to small random values between the range $[-0.5, 0.5]$. For the Backpropagation algorithm, a learning rate must be specified. It determines how weights are changed from successive iterations. In this work a small learning rate 10^{-3} is used. Another important feature to be configured in the ANN is the activation function for each network node. Each node j receives a set of inputs, $\{x_i\}_{i=1}^{n}$. The basic output of the node is given by the linear expression $o = x_1 w_{i1} + x_2 w_{i2} + \ldots + x_n w_{in}$. The activation function takes this expression as an argument and introduces a degree of nonlinearity that is valuable for most ANN applications. A typical function used is the sigmoid

$$\sigma(o) = \frac{1}{1 + e^{-o}}.$$

Regarding the optimal number of hidden nodes for the ANN model, Cybenko's theorem [12] proved that a feedforward ANN with any continuous sigmoidal nonlinearity and only a single hidden layer can approximate any continuous function. Given the choice of a layered feedforward network with one hidden layer, how many hidden neurons should be included? In general, networks with fewer hidden nodes are preferable as they usually have a better ability to generalize and less overfitting problems. But networks with too few hidden nodes may not have enough power to model and learn data. There is no theoretical basis for the selection of this parameter. Some attempts were made in the past but an appropriate a priori ANN configuration is still an open issue in machine learning [32]. In this work, the optimal number of hidden nodes is a key parameter in order to obtain well-trained ANNs able to generate artificial data logs. A number of different ANNs were trained for each one of the two TSs under study. Each ANN was configured varying the number of hidden neurons and the number of training steps. The following paragraphs explain how the right number of hidden neurons and training steps were chosen.

It is worth noting that the selection of training and testing samples may affect the performance of ANNs in production mode. The training sample is used for ANN model learning, whereas the testing sample is adopted for the evaluation of the model correctness (step 5b of CHROMUBE). Given that the size of the data logs is 20 days, the first 14 days were used for training (70%) and the last 6 days (30%) were devoted to testing (i.e. a holdout approach was followed for model evaluation).

Fig. 7 shows a set of points representing the evolution of the best generalization error (i.e. that obtained by using the ANN to make predictions from testing data) obtained when training and evaluating the network. Each point corresponds to an ANN with a different number of hidden neurons, from 2 to 35. Within the test process, a generalization error is generated by using the Mean Squared Error,

$$\mathrm{MSE} = \sum_{i=1}^{n} \frac{(y_i - o_i)^2}{n},$$

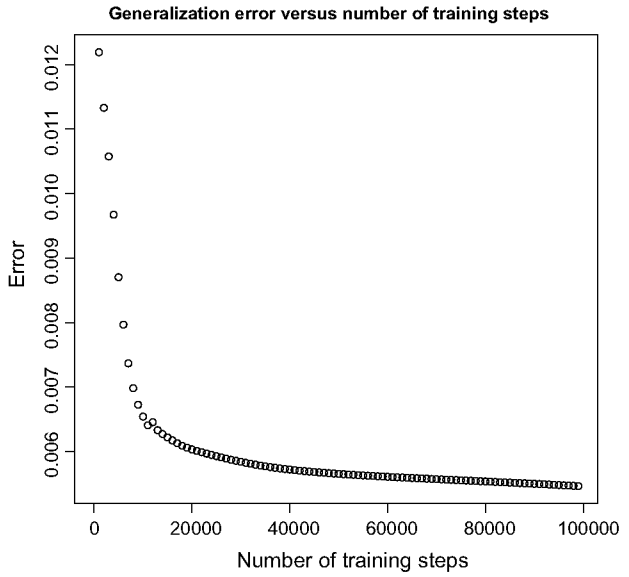


Fig. 8. Evolution of generalization error for a 100,000 training steps process.

where n is the number of evaluation examples, y_i are the output values for the i-th sample and o_i the network output for that sample.[4] Notice that the generalization error evolution represented in the graph clearly starts decreasing and then raises again resembling a parabola. Following this, configurations with 6, 8 and 10 hidden neurons were selected as the best candidate configurations.

Regarding the number of steps for training the ANN, there is a trade off that must be noticed. On the one hand, the higher the number of epochs used, the slower the training process is. On the other hand, longer training experiments in terms of epochs produce ANNs whose CMHBs are more accurate (see Section 4.3). But after a concrete number of epochs used in training a given ANN, the quality improvement of the model is not significant. However, there is no other way to test the ANN's quality as a CMHB than simulating it after training. Thus, we are forced to use different values for learning epochs in learning experiments.

Another important issue is illustrated in Fig. 8. It shows the evolution of generalization error for a concrete ANN with 8 hidden neurons trained until 100000 epochs (a similar phenomena occurs with the rest of topologies considered). The generalization error monotonically decreases no matter how many epochs are used. Thus, this reflects that the only relevant configuration parameter to control overfitting is the number of hidden nodes. In consequence, we tested values of 10,000, 20,000, 50,000, 100,000, 150,000 and 200,000 epochs. Section 4.3 shows how the right models have been achieved by using such bounds.

Section 4.4 offers more details about all the CMHBs obtained in each different learning experiment and also proposes validation and model selection methods whose objective is to select the most realistic CMHB.

4.3. Simulation of CMHB from a trained ANN

Let us suppose the ANN was successfully trained and tested (i.e. a correct behavior model was created). Now, it is explained how to generate from the ANN model a new log like the one appearing in Fig. 3, which represents the behavior of the simulated TS. In other words, how to perform a simulation of the CMHB. Given a tuple of location and timestamp as an input, the ANN computes an output $\vec{y} = (p_1, p_2, \ldots, p_r, ts)$ where every $p_k, 1 \leqslant k \leqslant r$ represents the probability of going to room k. But these outputs must be decoded in order to extract information directly interpretable as CMHB behavior, e.g. a pair (location, timestamp). Such decoding is performed by simulating the random process induced by probabilities $p_1, p_2, \ldots, p_r$. The simulation generates a concrete room. The value of ts represents how much time the subject is going to stay in the current room before moving to the next room, whatever it is.

The location chosen by this process represents the next location where the subject moves to once ts seconds are elapsed. The pair of location and amount of time obtained is used as the next input of the ANN, taking into account that the time entry results from adding the t value of the last entry and ts value of the ANN's prediction. The overall process is repeated iteratively. Each decoded output represents a log entry in the CMHB's behavior log which has the next form:

[4] MSE is adequate here, as well as many other error measures, because it is only used for model selection and not as an estimator of the ANN's quality when used in production mode. Thus, there is no special requirement for measuring the evaluation error.

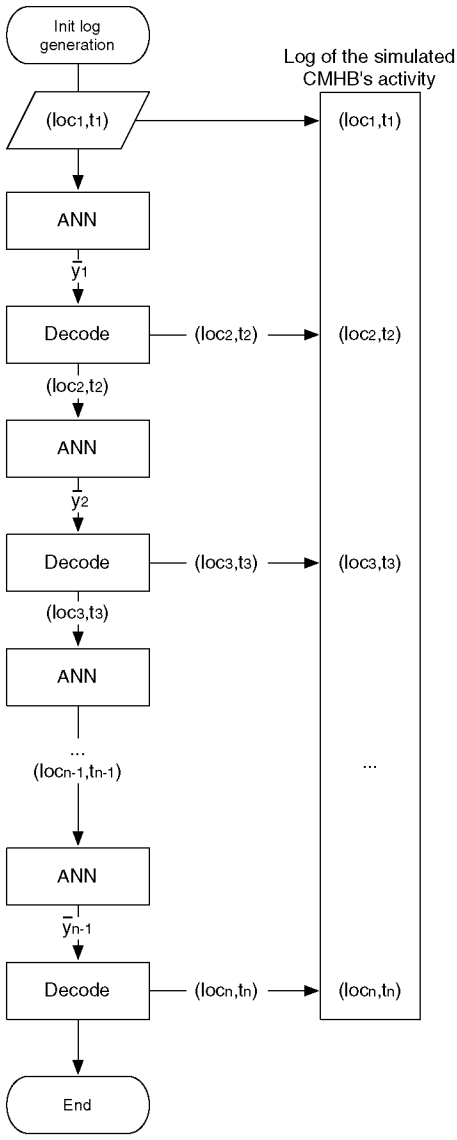


Fig. 9. Simulation process of a CMHB.

$$\{(loc_1, t_1), (loc_2, t_2), \ldots (loc_n, t_n), \}$$

where (loc_1, t_1) could be a tuple randomly selected from the test data set or any tuple chosen by the developer. The overall process is illustrated in Fig. 9.

4.4. Selection of the best CMHB

A number of different CMHBs were obtained following the steps introduced in the previous section. They were intended to represent the two TSs. In this section the CHROMUBE's approach for selecting the best CMHB for each TS is introduced. See Fig. 10 for a schematic representation of the process. There are three sequential tests a CHMB must pass to become a suitable model. First, the circadian rhythm must be checked. The subset of CMHBs that passes this first test is then statistically tested to check its similarity with the real TS. Finally, among those which are similar, the most similar CMHB to the TS is selected.

The first test is based on the ANOVA technique and its main purpose is to investigate if the CMHBs show regularity in their behaviors by following an approach used in Chronobiology: detection of circadian rhythms. A circadian rhythm is a biological rhythm of 24 h, and its detection is important to prove that the human evidences activity patterns during the day. In consequence, the human's behavior can be modeled. As described in [30], the ANOVA test [34] must be applied to the plexogram in order to detect circadian rhythms.

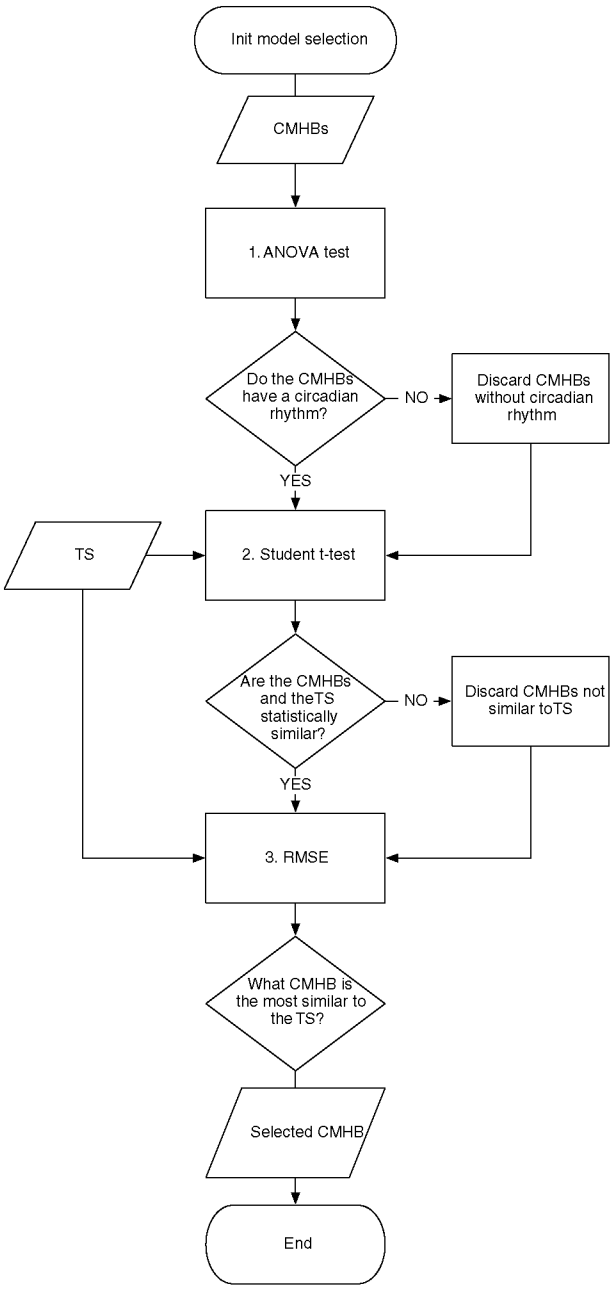


Fig. 10. Validation methods to select the best CMHB.

ANOVA applies a statistical test to determine whether or not the means of several samples are all equal. In this case study, a plexogram is partitioned in 24 intervals of 1 h each. Each of these intervals compounds a group of activity rates (i.e. as many values as the number of days devoted to each interval during the experiment), and their corresponding means (i.e. the points of the plexogram) are compared by ANOVA. Thus, the null hypothesis H_0, "*the mean of all groups are equal*", and the alternative H_1 are the following:

$$H_0 : p_1 = p_2 = \ldots = p_{24}, H_1 : p_1 \neq p_2 \neq \ldots \neq p_{24}.$$

If the main hypothesis is rejected, then the CMHBs pass the test as there exist different activity patterns in their behavior depending on the time of the day.

The second test is a statistical comparison of the original log data with the log data generated by the CMHB. Thus, a test that checks if two samples are highly similar is used, i.e. a Student's t-test for independent samples [34]. Let μ_{TS} be the mean

Table 1
ANOVA results for the CMHBs: p-values. $\xi_{A,B}$ refers to the CMHB for either TS_A or TS_B and h is the number of hidden nodes in the ANN.

	10,000 Epochs	20,000	50,000	100,000	150,000	200,000
$\xi_A,\ h=6$	7.43e−4	2.23e−4	4.6e−3	5.45e−4	2.37e−8	1.06e−5
$\xi_A,\ h=8$	6.70e−6	5.87e−5	5.42e−4	8.09e−6	1.2e−3	9.56e−5
$\xi_A,\ h=10$	1.49e−4	5.54e−3	1.28e−5	5.27e−3	1.49e−2	3.04e−3
$\xi_B,\ h=6$	4.12e−10	1.85e−5	4.88e−8	1.13e−10	2.72e−7	5.15e−7
$\xi_B,\ h=8$	4.09e−7	9.09e−9	1.47e−8	7.91e−9	9.04e−9	4.84e−7
$\xi_B,\ h=10$	1.68e−9	1.81e−8	1.44e−9	2.14e−9	2.64e−9	8.69e−9

Table 2
All p-values for Student's t test for statistical similarity between CMHB and TS. $\xi_{A,B}$ refers to the CMHB for either TS_A or TS_B and h is the number of hidden nodes in the ANN. Asterisks show the values which pass the test.

	10,000 Epochs	20,000	50,000	100,000	150,000	200,000
$\xi_A,\ h=6$	0.031	0.452*	0.104*	0.066*	0.007	0.187*
$\xi_A,\ h=8$	0.018	0.057*	0.513*	0.772*	0.181*	0.426*
$\xi_A,\ h=10$	0.134*	0.05*	0.062*	0.347*	0.857*	0.379*
$\xi_B,\ h=6$	0.001	0.013	0.008	0.126*	0.387*	0.01
$\xi_B,\ h=8$	9.8e−4	0.001	0.064*	0.657*	0.364*	0.972*
$\xi_B,\ h=10$	0.01	0.002	0.005	0.079*	0.52*	0.687*

Table 3
RMSE values for plexogram comparison between TS and CMHB. The asterisk shows the best CMHB for each TS.

	100,000 Epochs	150,000	200,000
$\xi_A,\ h=8$	3.869396	10.17856	7.199836
$\xi_A,\ h=10$	3.358413*	8.006409	5.62893
$\xi_B,\ h=8$	5.120109	6.496569	5.716604
$\xi_B,\ h=10$	5.378785	3.846136*	4.299864

of the sample composed of the points of the TS's plexogram generated from log data. Let also μ_ξ be the mean of the sample points from the plexogram generated from the CMHB's log data for the same TS. Then the tested hypotheses are the following

$$H_0 : \mu_{TS} = \mu_\xi, H_1 : \mu_{TS} \neq \mu_\xi.$$

If H_0 is rejected, H_1 must be accepted, thus the two samples are different.

After these two tests, if more than one CMHB are selected, a further test is done based on $\Delta(f_{TS}, \hat{f}_{TS})$. Let us suppose that f_{TS} generates a plexogram P_{TS} and $\hat{f}_{TS}$ generates a plexogram P_ξ, where ξ is a CMHB that imitates TS. The CMHB with the lower RMSE (Rooted Mean Squared Error) between the plexograms P_{TS} and P_ξ is chosen,

$$E(P_{TS}, P_\xi) = \sqrt{\sum_{i=1}^{24} \frac{(p_i - p_i')^2}{24}},$$

being p_i and p_i' the i-th point of P_{TS} and P_ξ, respectively.

4.4.1. Obtained results

The results derived from using the ANOVA test for every CMHB obtained appear in Table 1. As all the resulting p-values are lower than 0.05, all the CMHBs show a circadian rhythm. As a consequence, the ANOVA test does not reject any one of the CMHBs. All of them could represent a real human as they present circadian rhythms.

Results, i.e. p-values, for the second test are shown in Table 2. Values marked with asterisks pass the test, i.e. they are greater than 0.05 (95% confidence level). Notice that asterisks are more frequent as the number of epochs increases. If we focus on results with 8 or 10 hidden neurons and more than 50,000 training steps, none of the null hypotheses can be rejected. Thus, the CMHBs are statistically similar to the TSs.

Table 3 shows the results obtained by comparing the CMHB's plexograms with the TS's plexogram for both subjects. The best configuration is marked with an asterisk. The most similar plexogram to P_{TS_A} was obtained with 10 hidden neurons and 100,000 training steps. For P_{TS_B} the best option was 10 hidden neurons and 150,000 training steps. P_{ξ_A} and P_{ξ_B} for the best CMHBs are shown in Fig. 11.

The previous analysis is valid when the CMHB is intended to simulate the overall activity of the TS. However, a CMHB that behaves similarly to the TS in each room of the house may be needed. The next analysis shows that the ANN model is useful

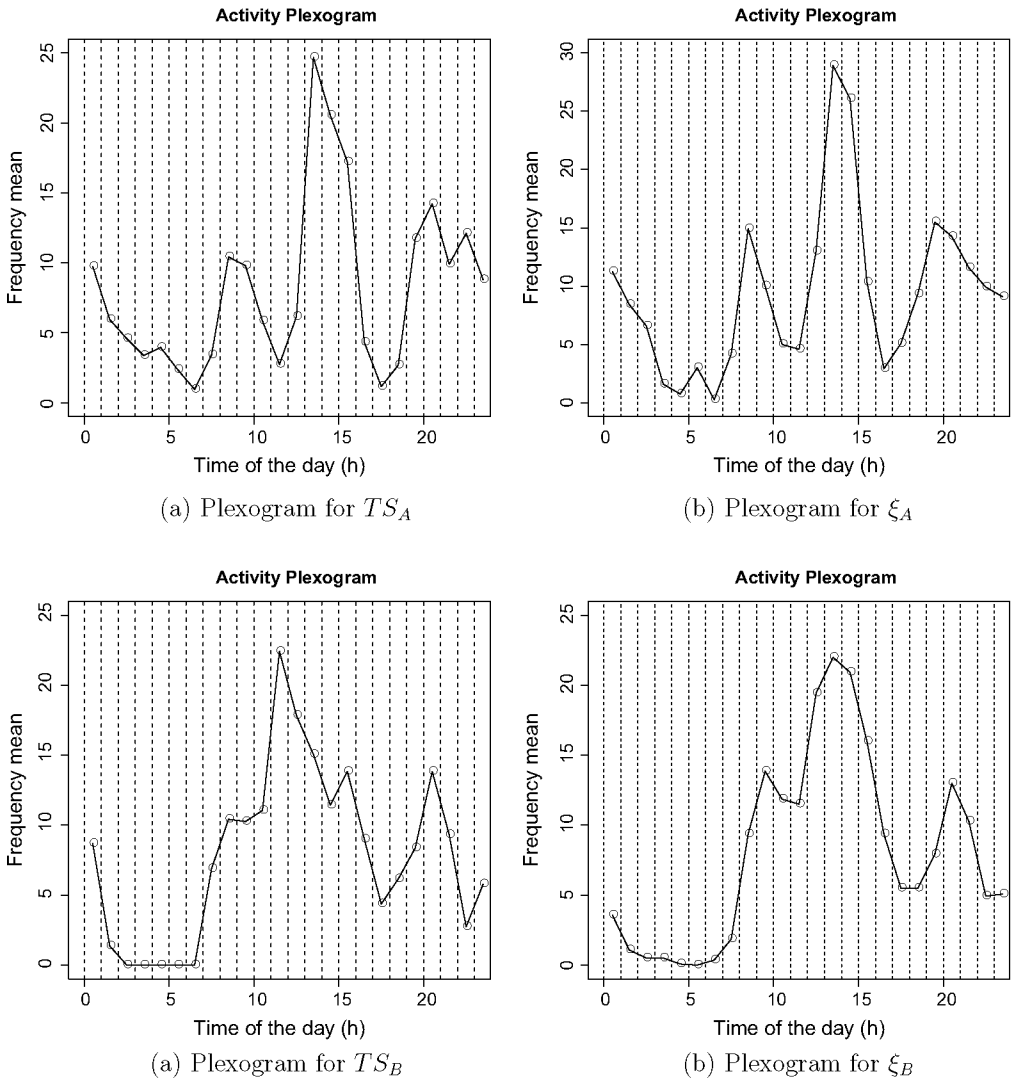


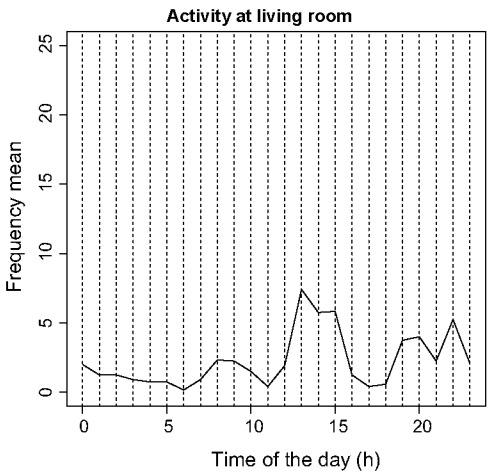
Fig. 11. Visual comparison between plexograms associated to (a) real subjects, (b) simulated subjects.

Table 4
RMSE values for by room plexogram of CMHBs.

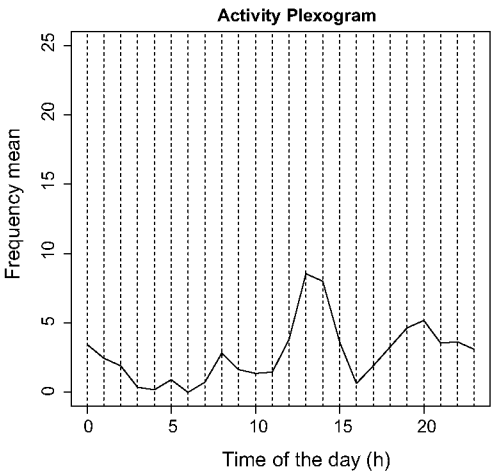
	Living room	Bathroom	Bedroom	Kitchen	Corridor
ξ_A	1.126671	0.4316742	0.5017331	1.299706	0.8198563
ξ_B	1.662312	0.9489902	1.192935	0.7046667	0.3079284

also to address this problem. Specific plexograms for each room of both TSs and CMHBs were generated and compared by the RMSE. Table 4 shows the error results obtained. Some plexograms are visually shown in Fig. 12. The artificial plexograms are actually similar.

Notice that the best ANN based CMHB for TS_A has a RMSE of 3.358513 (see Table 3). The best probabilistic automaton based CMHB for the same subject (see details at [9]) is 7.834205. It shows an improvement of 57.1%. For TS_B, the ANN based CHMB gets a RMSE of 3.846136 with respect to a RMSE of 5.232643 for the probabilistic automaton based CMHB, showing again an improvement of 26.5%. It is reasonable to expect that the CMHB's behavior based on a neural network model will be more similar to their TS's behavior than the one of the CMHBs previously obtained by the probabilistic automaton approach. In this sense, the new machine learning branch of CHROMUBE represents an improvement. This does not necessarily mean

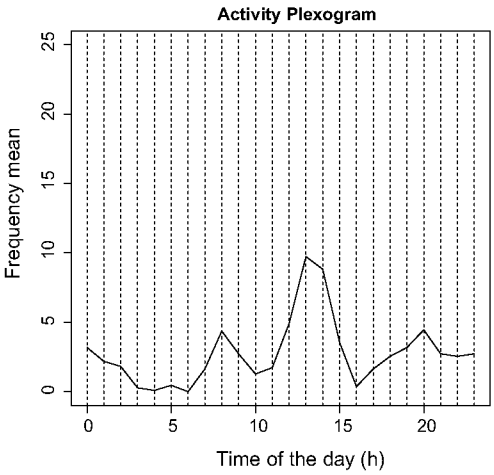


(a) TS_A at the living

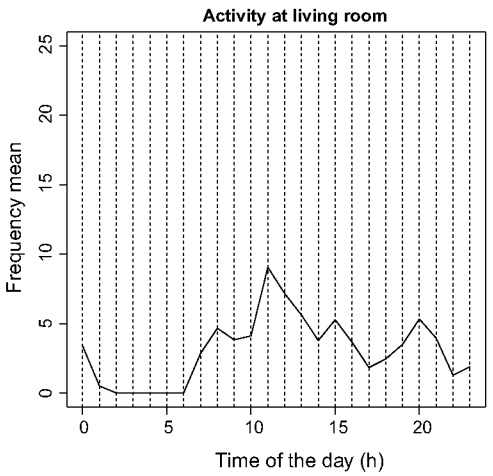


(b) ξ_A at the living

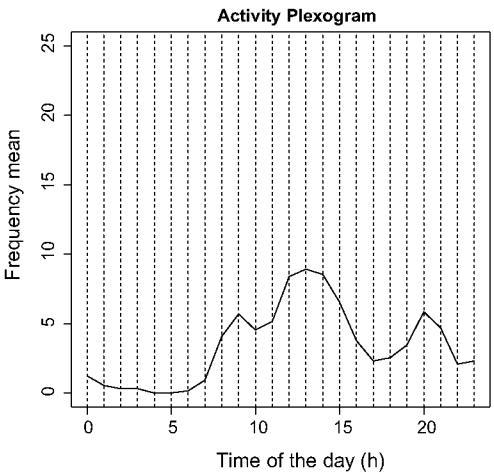
(a) TS_A at the kitchen



(b) ξ_A at the kitchen



(a) TS_B at the living



(b) ξ_B at the living

Fig. 12. Comparison between plexograms associated to (a) real subjects, (b) simulated subjects.

that the old branch should be removed. Depending on the data source, the TS and the validation scenario, a developer may prefer a manual configuration of the TS's behavior patterns.

5. Conclusions and future work

This paper presents an automatic process to create Computational Models of Human Behavior (CMHB) based on machine learning techniques for simulation. This process is an alternative extension of the CHROMUBE methodology. In CHROMUBE the CMHBs were originally created through manual analysis thanks to the use of Chronobiology techniques.

The proposal presented here allows for learning human behavior patterns automatically from sensor data and, therefore, avoiding the manual process which was necessary in the previous CHROMUBE proposal in order to configure the probability distribution functions of the probabilistic automaton.

With the aim of showing the proposal feasibility, it is presented an example which involves the use of artificial neural networks. This example reveals that machine learning techniques improve the results previously obtained by the generation of CMHBs. In fact, when data obtained from sensors are available, this process is able to generate more realistic human behaviors than those generated by a manual configuration of each target subject. This extension defines a more appropriate process when sensor data are available. It provides the system with a whole process of automatic generation of realistic CMHBs that are able to simulate human behaviors accurately.

In the future, CHROMUBE will be extended to supply it with the capability of detecting similar behaviors between different humans. This will enable the use of such behaviors in situations where the available data are not enough to generate an accurate model and to model multi-inhabitant environments more easily. Furthermore, optimization algorithms (e.g. evolutive algorithms) are being considered to obtain the most suitable configuration of the learning process parameters in a totally autonomous way (in the example presented, the number of hidden neurons of the ANN and the number of training steps). This may improve the quality of models and the usability of CHROMUBE by non machine learning experts.

Another possible future work implies the use of evolving system methodologies for supervised learning. Some methodologies in this direction can be found in [23,43]. These evolving systems modify their internal structure (e.g. the number of hidden neurons of an ANN) every time new input data are received. This way, it is possible to omit time-intensive retrainings on the complete data set. In the case of CHROMUBE, it could provide real-time learning from human behavior gathered every day.

Finally, another extension of CHROMUBE methodology for an easier and more automatic extraction of information and generation of models when sensor data are unavailable is being developed. Information comes, in this case, from experts on the behavior to be imitated. This extension will provide CHROMUBE methodology with a whole, easy and more automatic process regardless of the presence of sensors data.

Acknowledgement

This research work is supported by the Spanish Ministry of Economy and Competitiveness under the R&D project CALISTA (TEC2012-32457).

References

[1] Y. Al Mashhadany, Recurrent neural network with human simulator based virtual reality, Recurr. Neural Networks Soft Comput. (2012) 89–115.
[2] C. Atkeson, J. Hale, F. Pollick, M. Riley, S. Kotosaka, S. Schaul, T. Shibata, G. Tevatia, A. Ude, S. Vijayakumar, et al, Intelligent systems and their applications, IEEE 15 (2000) 46–56.
[3] M. Azam, J. Loo, S. Khan, M. Adeel, W. Ejaz, Human behaviour analysis using data collected from mobile devices, Int. J. Adv. Life Sci. 4 (2012) 1–10.
[4] L. Bao, Physical activity recognition from acceleration data under semi-naturalistic conditions, Ph.D. thesis, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2003.
[5] J. Barton, V. Vijayaraghavan, Ubiwise: a ubiquitous wireless infrastructure simulation environment, HP Labs, 2002.
[6] M. Bennewitz, W. Burgard, G. Cielniak, S. Thrun, Learning motion patterns of people for compliant robot motion, Int. J. Robot. Res. 24 (2005) 31–48.
[7] C.M. Bishop, Neural Networks for Pattern Recognition, Springer, 2006.
[8] B. Brumitt, B. Meyers, J. Krumm, A. Kern, S. Shafer, Easyliving: technologies for intelligent environments, in: Handheld and Ubiquitous Computing, Springer, 2000, pp. 97–119.
[9] F. Campuzano, J. Botia, A. Villa, Chronobiology applied to the development of human behavior computational models, J. Amb. Intell. Smart Environ. 4 (2012) 369–389.
[10] F. Campuzano, T. Garcia-Valverde, A. Garcia-Sola, J. Botia, Flexible simulation of ubiquitous computing environments, Amb. Intell.-Software Appl. (2011) 189–196.
[11] B. Cheng, Y. Tsai, G. Liao, E. Byeon, Hmm machine learning and inference for activities of daily living recognition, J. Supercomput. 54 (2010) 29–42.
[12] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. (MCSS) 2 (1989) 303–314.
[13] A.K. Dey, Understanding and using context, Personal Ubiquit. Comput. 5 (2001) 4–7.
[14] T. Duong, H. Bui, D. Phung, S. Venkatesh, Activity recognition and abnormality detection with the switching hidden semi-markov model, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, CVPR 2005, vol. 1, IEEE, 2005, pp. 838–845.
[15] I.O. Electrical, E.E. (IEEE), IEEE 90: IEEE Standard Glossary of Software Engineering Terminology, 1990.
[16] A. Fernández-Caballero, J. Castillo, J. Rodríguez-Sánchez, Human activity monitoring by local and global finite state machines, Expert Syst. Appl. 39 (2012) 6982–6993.
[17] M. Fritz, B. Schiele, Decomposition, discovery and detection of visual categories using topic models, in: IEEE Conference on Computer Vision and Pattern Recognition, 2008, CVPR 2008, IEEE, 2008, pp. 1–8.
[18] N. Gilbert, K.G. Troitzsch, Simulation for the Social Scientist, Open University Press, 2005.
[19] F. Halberg, F. Carandente, G. Cornelissen, G.S. Katinas, Glossary of chronobiology, Chronobiologia 4 (1997) 1.
[20] T. Hirzel, Visualizing exercise hidden in everyday activity, Ph.D. thesis, Massachusetts Institute of Technology, 2002.
[21] J. Hoey, P. Poupart, C. Boutilier, A. Mihailidis, Pomdp models for assistive technology, in: Proc. AAAI Fall Symposium on Caring Machines: AI in Eldercare.

[22] B. Jansen, R. Deklerck, Context aware inactivity recognition for visual fall detection, in: Pervasive Health Conference and Workshops, 2006, IEEE, 2006, pp. 1–4.
[23] N. Kasabov, Evolving connectionist systems: the knowledge engineering approach, second ed., 2007.
[24] S. Khan, M. Karg, J. Hoey, D. Kulic, Towards the detection of unusual temporal events during activities using hmms, 2012.
[25] B. Kröse, T. Van Kasteren, C. Gibson, T. Van Den Dool, Care: context awareness in residences for elderly, 2008.
[26] N. Landwehr, B. Gutmann, I. Thon, M. Philipose, L. De Raedt, Relational transformation-based tagging for human activity recognition, in: Proceedings of the International Workshop on Knowledge Discovery from Ubiquitous Data Streams, pp. 83–94.
[27] G. LeBellego, N. Noury, G. Virone, M. Mousseau, J. Demongeot, A model for the measurement of patient activity in a hospital suite, IEEE Trans. Inform. Technol. Biomed. 10 (2006) 92–99.
[28] J. Lester, T. Choudhury, N. Kern, G. Borriello, B. Hannaford, A hybrid discriminative/generative approach for modeling human activities, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence, pp. 766–772.
[29] L. Liao, D. Patterson, D. Fox, H. Kautz, Learning and inferring transportation routines, Artif. Intell. 171 (2007) 311–331.
[30] J. Madrid, A. Rol, Cronobiología básica y clínica, España: editec@ red, 2007.
[31] J.L. McClelland, D.E. Rumelhart, Explorations in Parallel Distributed Processing, MIT Press, 1988.
[32] D. Michie, D.J. Spiegelhalter, C. Taylor (Eds.), Machine Learning, Neural and Statistical Classification, Ellis Horwood, 1994.
[33] T. Mitchell, Machine Learning (McGraw-Hill International Edit), McGraw Hill Higher Education, 1997.
[34] D.S. Moore, The Basic Practice of Statistics, WH Freeman & Co., New York, NY, USA, 1999.
[35] G.J. Myers, C. Sandler, T. Badgett, T.M. Thomas, The Art of Software Testing, second ed., Wiley, 2004.
[36] H. Nait-Charif, S. McKenna, Activity summarisation and fall detection in a supportive home environment, Proceedings of the 17th International Conference on Pattern Recognition, 2004, ICPR 2004, vol. 4, IEEE, 2004, pp. 323–326.
[37] V.P. Niitamo, S. Kulkki, M. Eriksson, K.A. Hribernik, State-of-the-art and good practice in the field of living labs, in: Proceedings of the 12th International Conference on Concurrent Enterprising: Innovative Products and Services through Collaborative Networks, Milan, Italy, pp. 349–357.
[38] H. Nishikawa, S. Yamamoto, M. Tamai, K. Nishigaki, T. Kitani, N. Shibata, K. Yasumoto, M. Ito, UbiREAL: realistic smartspace simulator for systematic testing, in: UbiComp 2006: Ubiquitous Computing, 2006, pp. 459–476.
[39] E. O'Neill, M. Klepal, D. Lewis, T. O'Donnell, D. O'Sullivan, D. Pesch, A testbed for evaluating human interaction with ubiquitous computing environments, First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005, Tridentcom 2005, IEEE, 2005, pp. 60–69.
[40] D. Patterson, L. Liao, D. Fox, H. Kautz, Inferring high-level behavior from low-level sensors, in: UbiComp 2003: Ubiquitous Computing, Springer, 2003, pp. 73–89.
[41] M. Riedmiller, H. Braun, A direct adaptive method for faster backpropagation learning: the rprop algorithm, in: IEEE International Conference on Neural Networks, 1993, IEEE, 1993, pp. 586–591.
[42] B. de Ruyter, E. van Loenen, V. Teeven, User centered research in experiencelab, in: European Conference of AmI, LNCS, vol. 4794, Springer, Darmstadt, Germany, 2007.
[43] M. Sayed-Mouchaweh, E. Lughofer, Learning in non-stationary environments: methods and applications, 2012.
[44] E. Serrano, J.A. Botia, J.M. Cadenas, Ubik: a multi-agent based simulator for ubiquitous computing applications, J. Phys. Agents 3 (2009) 39.
[45] M. Vukovic, I. Lovrek, D. Jevtic, Predicting user movement for advanced location-aware services, in: 15th International Conference on Software, Telecommunications and Computer Networks, 2007, SoftCOM 2007, IEEE, 2007, pp. 1–5.
[46] M. Vukovic, G. Vujnovic, D. Grubisic, Adaptive user movement prediction for advanced location-aware services, in: 17th International Conference on Software, Telecommunications & Computer Networks, 2009, SoftCOM 2009, IEEE, 2009, pp. 343–347.