Genetic Programming and Evolvable Machines (2021) 22:101–135
https://doi.org/10.1007/s10710-020-09393-2

Fuzzy cognitive maps for decision-making in dynamic environments

Tomas Nachazel[1]

Received: 31 May 2019 / Revised: 29 March 2020 / Published online: 27 May 2020

Abstract
This paper describes a new modification of fuzzy cognitive maps (FCMs) for the modeling of autonomous entities that make decisions in a dynamic environment. The paper offers a general design for an FCM adjusted for the decision-making of autonomous agents through the categorization of its concepts into three different classes according to their purpose in the map: *Needs*, *Activities*, and *States* (FCM-NAS). The classification enables features supporting decision-making, such as the easy processing of input from sensors, faster system reactions, the modeling of inner needs, the adjustable frequency of computations in a simulation, and self-evaluation of the FCM-NAS that supports unsupervised evolutionary learning. This paper presents two use cases of the proposed extension to demonstrate its abilities. It was implemented into an agent-based artificial life model, where it took advantage of all the above features in the competition for resources, natural selection, and evolution. Then, it was used as decision-making for human activity simulation in an ambient intelligence model, where it is combined with scenario-oriented mechanism proving its modularity.

Keywords Autonomous systems · Decision-making · Dynamic environments · Fuzzy cognitive maps · Multi-agent models

1 Introduction

Fuzzy cognitive maps (FCMs) are powerful tools for the simulation of dynamic phenomena. They are generally used to predict or simulate systems that consist of many dependent variables in a complex dynamic structure [1]. FCM has proved to be a

Area Editor: Sebastian Risi.

✉ Tomas Nachazel
tomas.nachazel@uhk.cz

[1] Faculty of Informatics and Management, University of Hradec Králové, Rokitanskeho 62, 500 03 Hradec Králové, Czech Republic

strong decision-making method, even for autonomous agents. An agent is an intelligent computer system that is capable of evaluating a situation, making decisions and performing actions [2]. It is usually an entity with a location and an ability to move within the environment, but the proposed approach is also applicable to static ambient intelligence, in which the observed environment is inside the entity.

The motivation for this paper was the proposal of a new modification of the FCM with *Needs*, *Activities,* and *States* node classes (FCM-NAS) aimed at the decision-making of autonomous agents in a dynamic environment. The original design of FCMs does not allow for differences between concepts; each concept node has the same range of values, the same behavior, and the same interpretation. This approach is sufficient in a dynamic system with equal elements of the same type; however, some changes are unavoidable if an FCM needs to process some concept nodes differently. The management of different kinds of nodes after computation often becomes very confusing and context-dependent, hindering the modularity and scalability of the FCM. This paper offers a new design of FCMs that allows for various types of nodes. The proposed method processes nodes according to an implemented classification during computation, meaning that special treatment after computation is not necessary.

The FCM-NAS approach is quite different from the classical FCM in terms of its structure and use. This method has several advantages that are not possible for a standard approach to achieve without ad-hoc editing of the algorithm. For instance, it enables decision-making, the easy processing of input from sensors, faster system reactions, more realistic behavior in simulations (disabling parallel activities if necessary), the simulation of inner needs, adjustments to the frequency of computations in a simulation, and self-evaluation of the agent (fitness), which supports learning.

To verify the proposed design and demonstrate its abilities, this paper describes the implementation of the design within two models: an artificial life model and an ambient intelligence model. The first model was chosen for its ability to test the quality of artificial intelligence, as thousands of test subjects with various attributes and behaviors compete in a single model. Artificial life modeling allows us to explore real natural phenomena, emergence, and evolution within a runtime of only minutes. Autonomous individuals behave according to their needs and the situation in their vicinity, and their intelligence is tested through competition for limited resources. To survive, they also need to deal with changes in a dynamic environment. Natural selection guides the main directions of evolution and the specialization of various species after several generations. The second model, the ambient intelligence model, was chosen to demonstrate a combination of FCM-NAS and scenario-based behavior, which proves its ability to cooperate with other decision-making mechanisms effortlessly.

The proposed concept has already been modified for large-scale models in which performance is a key factor. In the latter case, an analytic hierarchy process replaced the decision-making part of FCM-NAS [3]. The analytic hierarchy process combination improved performance, but the quality of decision-making and possibilities for evolutionary progress were slightly decreased. Although this modification has already been published [3], a full description of FCM-NAS has not previously been published, which is the objective of this paper.

The paper is organized as follows: Sect. 2.1 introduces the essential elements of the original FCMs. Section 2.2 provides an overview of the various uses of FCMs, and the extensions and modifications that support autonomous agents or systems. Section 3 continues with a description of the new FCM extension: FCM-NAS. This section covers the fundamental aspects of the modification and is followed by Sect. 4 in which some more advanced optional additions are described. Section 5 presents two example use cases of this method. Firstly, it provides the complete, step-by-step design of the FCM-NAS for an artificial life model and discusses the testing of the created FCM-NAS in the model. Later, this section also describes the implementation of the proposed approach into an ambient intelligence model to prove its modularity. Finally, Sect. 6 gives concluding remarks.

2 Related work

2.1 Fuzzy cognitive maps

FCMs are dynamic systems of concept nodes with a complex network of relations. The values of the concept nodes change through iterations according to a set of relationships. The original FCMs are based on a collection of n concept nodes C, relations w_{ij} between these nodes and a function used to adjust values of nodes. An FCM consisting of a graph with directed, weighted edges can be represented as an adjacency matrix. The adjacency matrix R in Eq. (1) is a commonly used form to express behavior through iterations [1].

$$R = \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nn} \end{pmatrix} \tag{1}$$

where n is the number of concept nodes. Each value w_{ij} represents the influence of node c_i on node c_j. Throughout this paper, the indices i, j take values in $\mathbb{N}$ to refer to a specific node. Values w_{ij} may be any real number between -1 (strong negative causality) and 1 (strong positive causality):

$$\forall w_{\mathrm{ij}} \,:\, w_{\mathrm{ij}} \in [-1, 1] \tag{2}$$

If $w_{ij} = 0$, then node c_i has no direct influence on node c_j. If $w_{ij} > 0$, then the larger the value of node c_i, and the more it raises the level of node c_j. If $\mathrm{w}_{ij} < 0$, then the larger the value of node c_i, and the more it lowers the value of node c_j.

In addition to the matrix R, an FCM needs the truth values of nodes based on fuzzy logic [4]. Let v_i denote the truth value of node c_i. This value represents how active or strong its corresponding concept node is. For example, an agent with a high level for the value corresponding to concept node *Fatigue* means that the agent is tired. Names of nodes determine the understanding and design of concept nodes and their relationships. When designing causal relations, each concept is interchangeable with its opposite counterpart after inverting all the relations of the inverted

concept; then, this change does not influence the logic of the FCM [1] (e.g., the relation '*Fatigue* positively influences *Sleep*' is equal to '*Fatigue* negatively influences *Vigilance/Not-Sleep*').

Since this paper anticipates that an FCM is used in a dynamic environment, it will be changed through iterations (often called time steps in simulations). A time variable is therefore necessary for calculations. Let us denote time steps with t, which is a positive whole number starting at zero as FCMs calculates values on a discrete time scale and is only able to advance in time. Note that simulation may run in continuous time; in that case, an FCM is processed at regular intervals corresponding with the time unit for which the FCM was designed. Changing values of nodes are then specified as v_i^t, which is the value of concept node c_i at time step t. As a truth value, v_i^t is always a real number ranging between 0 (definitely not true) and 1 (definitely true). Even if the value exceeds these limits after computation, it needs to be immediately reduced to the valid range:

$$\forall v_i^t \, : \, v_i^t \in [0, 1] \tag{3}$$

These values form a vector V^t; a one-dimensional array also valid only for the time step t:

$$V^t = \begin{pmatrix} v_1^t \\ v_2^t \\ \vdots \\ v_n^t \end{pmatrix} \tag{4}$$

The sizes of all components are constant since concept nodes are neither removed nor added. The vector V^t is updated at every time step, and the content of R is static. Equation (5) shows the computation of iteration at time step t based on the original definition model, and Eq. (6) is a version in which the product of the multiplication is added to the previous value, known as the incremental model [5].

$$V^t = f\left(R \cdot V^{(t-1)}\right) \tag{5}$$

$$V^t = f\left(V^{(t-1)} + R \cdot V^{(t-1)}\right) \tag{6}$$

In both equations, the function f represents a transformation of values. It is a real function of a real variable. Many diverse types of nonlinear functions can be used (e.g., sigmoid, hyperbolic, step). The primary task of this function is to keep the values within the interval from zero to one. The model described in this paper mostly uses a simple linear transformation unless a value exceeds this interval:

$$f(x) = \begin{cases} 0 \; for \, x < 0 \\ x \; for \, x \in [0, 1] \\ 1 \; for \, x > 1 \end{cases} \quad where \, x \in \mathbb{R} \tag{7}$$

The original FCMs handle all concept nodes in the same way, which causes a few issues in systems that contain elements of a different type. For example, if an FCM considers a number of inputs and then decides whether to trigger an action, then a trigger node has to be processed differently to the other nodes. In addition to choosing a sufficient level of the node to trigger an action, it usually has to recognize only two states of node: an action is performed or not. Such differences in the processing and interpretation of specific nodes make FCMs generally confusing and less modular, because they create arbitrary links between the implementation of the method and designed behavior. This is why a modification which is proposed in this paper to handle such situations should be used.

2.2 Uses and extensions of FCMs

FCMs offer a powerful approach to modeling various systems that consist of many dependent variables within a complex structure. Usually, an FCM is designed for a single, narrowly focused task, such as a support tool for analysis [6, 7], decision-making [8, 9], predictions [10], or various tasks in social sciences [11, 12]. These FCMs aim to generate specific values or a steady state after a few iterations. However, this paper defines an FCM as a decision-making method for intelligent systems in dynamic environments. Those systems need to handle a much wider range of situations, and require a slightly different approach to FCMs based on the core mechanics, but at the same time supporting decision-making, goal-oriented behavior, or machine learning. FCMs with a few enhancements have been shown to be strong artificial intelligence methods, even for autonomous agents [13].

In the field of intelligent systems, some projects aimed at interaction with humans use an FCM to simulate emotions. A virtual pet was designed for educational purposes, based on an FCM that ensures believable reactions to the user's actions [14]. An ambient intelligence system also enriched its user interface with FCM-based emotions to provide additional comfort and to respond naturally to the presence of users [15].

In many projects, FCMs are a core component of a system or agent in a model. For instance, this approach has been used by monitoring systems in which an FCM assesses risk in critical situations [16], a situation awareness model for infantry platoon leaders [17], and even landing site selection for planetary exploration [18].

2.2.1 Adaptive FCMs

Several articles have addressed the possibilities of learning FCMs [5, 19]. They are mostly focused on the design of FCMs, where a learning algorithm helps to prevent human errors during the design, or approximate missing or unclear data. Especially in projects with a dynamic environment, where FCMs need to deal with continuous changes, unsupervised training of the FCM or even its adjustment during runtime are desirable features. According to [5], genetic algorithms and Hebbian algorithms are commonly used as bases for various extensions to adjust the configuration of FCMs.

A dynamic fuzzy cognitive map (DFCM) is an extension focusing on the adaptation of FCMs. It is based on the reinforcement learning of a random neural model,

which was designed to react to random events by modifying causal relations. The main feature of this approach is its ability to change the weights of relations of an FCM during runtime, enabling adaptation and changes in behavior at runtime. Although its use is suitable for autonomous agents and systems, it does not necessarily overlap with the modification proposed in this paper because it focuses rather on the learning process of the system instead of its general structure and processing. This extension has been used as a supervision system [20], a navigation system for a robot [13, 21], and an intelligent controller [22]. In the last case, the authors tested a DFCM on a sample industrial mixer process. The authors set boundaries for some concepts and let a DFCM adjust weights (i.e., influences) between the concepts to maintain levels within specified limits. As the authors state in [13], the main disadvantage of DFCMs is their complexity even for small systems which hinders a manual development of the model.

In [23], authors evaluated their extended idea of DFCM: Dynamic Rule-based Fuzzy Cognitive Maps (DRBFCM). It uses fuzzy rules to adapt weights during a simulation through a set of fuzzy *IF-THEN* rules expressed by experts in the modelled field. This rule set helps with designing the model, as it does not require the precise recognition of weights because DRBFCMs adjust weights dynamically. An experiment reported in [23] made better predictions with a DRBFCM than the original FCM.

Several papers describe the mechanisms of learning by evolutionary principles [24]. In [25], multi-agent genetic algorithms were used to train an FCM. In a grid layout, agents (i.e., their genes) compete with each other by generating values. The values are then evaluated by their similarity to the desired solution. The goal of this approach is to find weights of the FCM's causal relations that would make its response as close to the observed data as possible. This paper [25] demonstrated the abilities to efficiently teach an FCM to capture the causal relations of the modeled problem. The method uses a simplified multi-agent approach as a tool for learning with its purpose to reconstruct time series, which differs from our proposed approach focusing on the decision-making of agents in a complex dynamic environment.

2.2.2 FCMs in multi-agent systems

The connection of FCMs with multi-agent systems provides some beneficial features. In [26], the authors propose a hybrid approach focused on influencing agents' mental models: *CoFluences*. This approach assumes that every agent has a different configuration of an FCM (i.e., a different view of the modeled problem) and agents directly influence each other. Some nodes are able to influence or be influenced by other agents. *CoFluences* use FCMs in a classic way as a mental model of a static problem. Agents influence each other by direct virtual links between their FCMs, which is still very different from the approach proposed in this paper, where each agent is a separate entity able to interact with the environment and other agents only through its sensors and actuators.

As described in [27], there are two approaches to incorporating an FCM into multi-agent systems: an FCM representing a macro view of the whole model, or FCMs on a micro level as a property of each agent. Narrowing the group of possible uses to decision-making, these two approaches translate into two areas of

multi-agent systems that are suitable for FCMs: a supervisor monitoring or controlling a whole model [20, 28], or an artificial intelligence controlling individual agents [29, 30]. The latter is not widely used, since FCMs in their original form are not convenient for this purpose, and modifications that could support this role are not yet available; however, many of the works mentioned above touched at least slightly on the problem of FCMs for autonomous systems. Moreover, some models already process agents with an FCM, such as the artificial life model *ALModel*, the design of which forms part of this paper (more in Sect. 5), and the *EcoSim* model [30].

EcoSim, an artificial life model, uses an FCM to process the behavior of individuals. The simulation allows for the evolution of values in the FCM, which enables the adaptation of behavior. Individuals can choose from a limited set of basic actions and select the optimal one for the current situation. The development and behavior of the population emerge from interactions between agents. *EcoSim* contains two types of species, predator and prey, which both evolve to increase their chances of success against the rival species. The decision-making of both the prey and predator species differs in terms of its actions and observed properties. High-level and low-level control are combined in a single map, which generates a very complex FCM (26 concepts). Despite this complexity, the behavior is focused only on reproduction and the management of food and energy. The patterns generated in this way are visually close to those of cell-based models. *EcoSim* aims to observe the emergence and evolution of the population, rather than to simulate the behavior of individuals realistically [31].

3 Fuzzy cognitive maps with needs, activities and states

This section presents a new approach to design FCM-NAS, FCMs with three node classes: *Needs*, *Activities* and *States*. It makes the creation of FCMs for more complex systems with various types of concept nodes more comprehensive, effective, extensible and systematic. This new proposed extension of FCMs offers several advantages that would be impossible to achieve with the general classical approach. Each concept class uses a different computational model or transformation function which later allows additional useful features. This section describes the overall structure of the extension in the following subsections, and examines individual classes and the specifics of their computation process in detail.

3.1 Structure

The fundamental step required to introduce FCM-NAS into a project is the partition of concept nodes into three classes. $\mathbb{C}$ is a partition of the original set of concept nodes C (see Eq. (8)). C_N denotes the set of nodes which were identified as *Needs*, C_A represents the set of *Activities*, and C_S is the set of *States*:

$$\mathbb{C} = \left\{ C_N \cup C_A \cup C_S \right\} \tag{8}$$

Since classes are pairwise disjoint sets, each concept node is in exactly one class. The original notation of the number of general concept nodes n is extended to distinguish the number of concepts in new classes: n_N for the number of *Needs*, n_A for the number of *Activities*, and n_S for the number of *States*. The following statements then result from the features of the new structure:

$$\forall c_i : c_i \in \left(C_N \cup C_A \cup C_S\right) \tag{9}$$

$$n = n_N + n_A + n_S \tag{10}$$

The notation of individual nodes (c_i) and their values (v_i^t) remains the same as in the original FCM design. To distinguish the assignment of nodes for the introduced classes, it is necessary to define the collection of nodes as having a static order of nodes starting with *Needs*, then *Activities*, and ending with *States*. Nodes are thus uniquely identified based on their index:

$$c_i \in \begin{cases} C_N \ for\, i = \{1, \ldots, n_N\} \\ C_A \ for\, i = \{n_N + 1, \ldots, n_N + n_A\} \\ C_S \ for\, i = \{n_N + n_A + 1, \ldots, n\} \end{cases} \tag{11}$$

3.2 Concept class needs

Needs are the first class of concept nodes. Unlike other classes, *Needs* keep their previous value as their starting point in iterations, meaning it is the only class whose values are developing through time instead of calculating it every iteration from scratch. A designed system usually has to observe a variable and perform an action repeatedly to keep the variable under control. If the action costs limited resources or time, the system should consider those costs, as it may not be efficient to take actions too often. The system could then use sensitive balancing or a time delay which *Needs* allow. Alternatively, systems have at least one purpose, which it is trying to fulfill, and a measure of success might be useful. In such cases, *Needs* are the optimal choice for the representation of the concepts. It is an obvious choice for the simulated biological needs of agents in an artificial life model. However, it is crucial even in non-biologically related areas: it can be used for a simulated level of satisfaction with memory management in a system, where its corresponding action repeatedly moves data from temporary fast memory to persistent database, or a set of needs balancing performance or workload between available resources.

The main difference between *Needs* and the other classes is their behavior during computation. Their computation is based on the incremental model introduced in Eq. (6), meaning it keeps its previous value and adds (or subtracts) an increment based on the influence of other nodes. If there are no active influences from the other nodes, then it holds its value. Equation (12) shows the computation of value v_i^t of node c_i. This equation uses the basic transformation function

f [see Eq. (7)]. As a reminder, note that the value w_{ij} represents the influence of node c_i on node c_j.

$$v_j^t = f\left[v_j^{t-1} + \left(\sum_{i=1}^{n} w_{ij} \cdot v_i^{t-1}\right)\right] \quad for\ j = \{1, \ldots, n_N\} \tag{12}$$

The value of a *Need* node represents the level of the necessity to do something to satisfy the corresponding need. Zero means the need has been satisfied, and an agent does not have to do anything. If its level approaches one, then the agent has to satisfy the need and should take appropriate action as soon as possible. The designer determines the threshold level of a *Need* node, which triggers a corresponding activity. This design choice depends on the configuration of relations in matrix R. The design of *Needs* and their integration into FCM-NAS enable features like the adjustable frequency of computations in a simulation, the addition of true positive causality to itself, the self-evaluation of an FCM-NAS, and the varying necessity of actions (all described in Sect. 4).

3.3 Concept class activities

The next class of concept nodes, *Activities*, represents all possible actions that a system or agent can perform. If an agent with an FCM-NAS is not just a passive observer and has to react, manage, or in any way affect its environment or itself, then there are two possible solutions. The first approach involves another mechanism outside of an FCM that reads values from the FCM and makes a decision [3]. The other approach places the actions directly inside the FCM-NAS, which then holds the decision-making responsibility.

An action either takes place or it does not, thus after computation, the *Activities* only have two possible values: zero (*false*; the action is inactive) or one (*true*; an agent performs the activity). During their computation, however, these values retain the full interval from zero to one. They are calculated with a similar algorithm as general nodes in the original FCMs [based on the definition model introduced in Eq. (5)]. Equation (13) shows the full computation of *Activities*.

$$v_j^t = f_a\left(\sum_{i=1}^{n} w_{ij} \cdot v_i^{t-1}\right) \quad \text{for} j = \{n_N + 1, \ldots, n_N + n_A\} \tag{13}$$

The values are rounded using a simple algorithm (the transformation f_a), which decides the activation of *Activities* based on their truth values acquired from the computation. Depending on whether parallel activities are available, it selects only the activity with the highest value (see Sect. 4.3) or performs all activities reaching a certain critical level a_c as seen in Eq. (14):

$$f_a(x) = \begin{cases} 1 & if\ x \geq a_c \\ 0 & otherwise \end{cases} \quad where\ x \in \mathbb{R} \tag{14}$$

In cases when *Activities* are supposed to provide a truth value to express an intensity of actions, basic f transformation Eq. (7) could be used. Transformation functions are adjustable to the demands of the system without any issues with the rest of the design.

Generally, FCMs are not suitable for a combination of high-level decision-making ("what should be done") and low-level operations ("how it should be done"; e.g., pathfinding). That combination often requires too many variables in one structure which may cause performance issues in large scale models as the time complexity of FCMs is $O(n^2)$ [3].

3.4 Concept class states

The third class of concept nodes *States* is very similar to general concepts in the original FCMs. The way it is calculated is almost the same as for the original definition model [see Eq. (5)], and its purpose does not have a narrow focus, as in the previously presented classes:

$$v_j^t = f\left(\sum_{i=1}^{n} w_{ij} \cdot v_i^{t-1}\right) \quad for\, j = \left\{n_N + n_A + 1, \ldots, n\right\} \tag{15}$$

Besides general concepts, *States* are an advantageous choice for external input nodes. If an agent needs to be able to perceive an attribute of the environment and take it into account during the decision-making process, then it requires a dedicated *State* node in its FCM-NAS. As a property of the environment, the value of this node is not directly affected by any node in the FCM-NAS. All relations to this node in matrix R therefore equal zero, which allows its whole calculation to be omitted. Instead, it is updated by sensors. Obviously, the inserted values have to be transformed into truth values (ranging from zero to one).

4 Additional extensions to FCM-NAS

While the previous section described the core of the proposed method, this section provides its optional extensions that are very useful for certain systems but not necessary for others. Although a few of these are dependent on each other (the dependencies are noted), the core design presented above can be implemented independently of the following options. Note that more extensions can always be proposed as this is not an exclusive list.

4.1 Frequency of computations

Since agents process an FCM-NAS through iteration in discrete time steps, the selection of the period of virtual time between computations of FCMs is a crucial

issue, regardless of whether the system deals with continuous real-time or discrete time steps. The values of an FCM in a dynamic environment are always valid only within the particular interval for which the FCM was designed. Figure 1 shows how an FCM-NAS perceives a continuous variable with different settings for the frequency in continuous time. A FCM-NAS with a frequency of 0.5 is four times more demanding in terms of performance but is also more precise than one with a frequency of 2.0. The parameter g is a multiplier representing the length of an interval between computations (measured in virtual time), which is relative to the default length of the interval for which a model was initially designed.

In a model with continuous time, the designer chooses how often an FCM updates its values. In a model with discrete time, the problem instead lies in deciding how much virtual time (or how many time steps) elapses between the computations or how much the environment changes in a single time step. After an FCM is designed, any change to the length of this interval invalidates certain values related to dynamic phenomena in the environment. With the proposed solution, however, the effect of the frequency of computations on concept classes is evident: only *Needs* are affected as they are the only time-dependent class of nodes. *Activities* are not, since a decision could be made at any time. *States* typically are not affected, but since they have a broader use, some may be affected in cases when a node has non-zero relation with itself.

The effect of this frequency on *Needs* causes differences in their growth (or decrease). This can be easily compensated for with a simple enhancement of the calculation, and the nodes always adjust to the current simulation speed. Equation (16) shows the addition of the parameter g, which adjusts the size of the increment every time step.

$$v_j^t = f\left[v_j^{t-1} + \boldsymbol{g} \cdot \left(\sum_{i=1}^{n} w_{ij} \cdot v_i^{t-1}\right)\right] \quad for\, j = \{1, \ldots, n_N\} \tag{16}$$

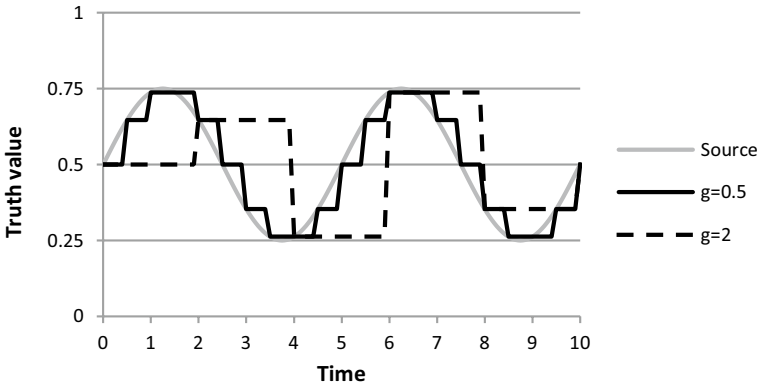

Fig. 1 Comparison of different settings of frequency of computations

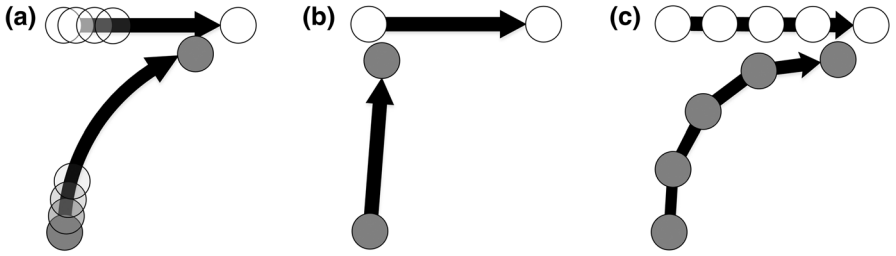

Fig. 2 Example of different behavior based on the frequency of computation; the dark dot represents predator following a prey (white dot) in **a** continuous time, and model with **b** $g = 1$ and **c** $g = 0.25$

The values of the *Needs* reach $(1/g)$ times more computations per virtual time unit than they would have with $g = 1$. Figure 2 shows an example in an artificial life model. A predator would reach his prey after one time step with the default frequency, but if the prey moved, then the predator missed it. With the frequency at 0.25, the predator processes computations four times. Of course, in both cases, it can run the same length for the same amount of virtual time, but due to the frequency of computations, it has four times as many opportunities to reconsider the activity or the direction of its movement.

The main advantage of this feature is the adjustability of a model during its development. It allows a designer to set values according to a specific time frame during a design phase and later adjust the time frame without the requirement of redesigning the whole matrix of relations. For example, if the value of node c_1 (for simplicity, assume it is always at 1.0) is supposed to increase the value of the *Need* node c_2 at rate 0.6 per hour, and the matrix is designed in this time-scale in mind, then w_{12} would be set at 0.6. Later in the development, a designer would realize that one computation per hour is not enough for the system. For the period of one minute between computations, parameter g would be set at 1/60, and the system would process FCM once per minute. Thanks to this feature, the designed increase rate of *Need* node c_2 remains automatically at 0.6 per hour (or 0.6/60 per minute). Otherwise, the designer would need to edit every value in matrix R by hand with every change to the frequency of computations of the model. This process would be especially laborious when testing different computation intervals.

This is very similar to sampling in signal processing; when samples are too far apart, much of the information between them can be lost; when samples are too close to each other, its processing is more demanding. Generally, a shorter interval generates better reactions of agents, but it raises performance demands of simulations as it is processed more often per virtual time unit. Note that the frequency parameter of agents may vary in a single system, meaning that they can have different reaction times.

4.2 Faster reactions of agents

As seen in Eqs. (5) and (6), the original FCMs compute the current values by using the values of the previous iteration. This procedure inevitably causes a delay between

a stimulus and the corresponding reaction (i.e., one time step or the interval between computations of the FCM). Depending on how often the FCMs are recomputed, this delay may cause problems if the short reaction time is essential for the proper operation of a system. For example, a monitoring system should react to fire immediately, as soon as sensors detect it, rather than waiting for the next iteration to take action.

Some researchers solve this with incremental or cumulative models [5]; however, their goal is to achieve FCM with stable converged values in the shortest time possible. FCM-NAS does not seek one set of values in order to decide; instead, it uses the values to make decision at every time step (i.e., every iteration is the solution to the current situation). Considering the objectives and abilities of this method, there is a far more efficient way to ensure not only faster but even instantaneous reactions of agents.

Thanks to the partition of concept nodes to different classes, the computation of an FCM-NAS can be divided into three parts, which can then be performed in any order. Some parts can even consider the values of the current iteration from parts that have already been computed. The best order has proved to be as follows: first the *Needs*, then the *States* and finally the *Activities*. This is because *Needs* do not have to correspond to the most current values; since they use an incremental model as shown in Eq. (12), their values are primarily based on their own previous values and actions performed in the previous time step. *States* may be based on the current values of needs, but also contain external inputs that have to be considered in decision-making as soon as possible. Finally, *Activities*, as the decision-making part of the model, should access the latest values in order to give the best possible reaction to the current situation.

The implementation of this feature requires only the replacement of values from the previous time step v_i^{t-1} with the current ones v_i^t for classes that have been already computed. Equations (17) and (18) show adjusted expressions for the order recommended above. The equation for *Needs* is not affected since no other current values are yet available for time step t.

$$v_j^t = f\left[\left(\sum_{i=1}^{n_N} w_{ij} \cdot v_i^t\right) + \left(\sum_{i=(n_N+1)}^{n} w_{ij} \cdot v_i^{t-1}\right)\right] \quad for\, j = \left\{n_N + n_A + 1, \ldots, n\right\} \tag{17}$$

$$v_j^t = f_a\left[\left(\sum_{i=1}^{n_N} w_{ij} \cdot v_i^t\right) + \left(\sum_{i=(n_N+1)}^{(n_N+n_A)} w_{ij} \cdot v_i^{t-1}\right) + \left(\sum_{i=(n_N+n_A+1)}^{n} w_{ij} \cdot v_i^t\right)\right] \quad \text{for } j = \left\{n_N + 1, \ldots, n_N + n_A\right\} \tag{18}$$

For example, in an artificial life model, agents have two concept nodes: the *Danger* state and the *Escape* activity. When an agent recognizes a dangerous situation, it should immediately escape rather than wait until the next time step to take action. Tables 1 and 2 show both approaches during a situation with a predator appearing in an agent's vicinity at time step $t=2$. While the agent with the original approach reacts by escaping at time step $t=3$ (one step after noticing danger), the agent with FCM-NAS using faster reactions starts the escape at the same time step as noticing a predator ($t=2$) because it uses the updated values for *Activities* (i.e., decision-making). Obviously, an agent with the FCM-NAS with this feature has a much better chance of escaping and surviving.

Table 1 The delay in reaction time in the original FCM

Time step	1	2	3	4	5	6	7	8
Danger	0	**1**	0	0	**1**	**1**	**1**	0
Escape	0	0	**1**	0	0	**1**	**1**	**1**

Table 2 Immediate reactions in the FCM-NAS

Time step	1	2	3	4	5	6	7	8
Danger	0	**1**	0	0	**1**	**1**	**1**	0
Escape	0	**1**	0	0	**1**	**1**	**1**	0

4.3 Disabling parallel activities

In many systems, the individual actions are independent of each other; however, there are cases where an agent is limited to one action per time step since different activities require the agent to be in different locations or several of them utilize a single actuator. The original FCM method cannot restrict this without another algorithm that processes and adjusts the values of specific nodes. If any process or value is bound to specific nodes by its position in an FCM, then the algorithm has to be adjusted after any change. This is confusing and a less modular approach.

The FCM-NAS uses a simple adjusted transformation algorithm for its *Activities*, meaning that it is not bound to specific nodes, which allows simple changes in concepts without disrupting the algorithm. This feature is useful in human activity simulation, artificial life models, or more generally for any non-trivial decision-making. For example, as a simulated person, an agent should not be able to eat, drink, and sleep at the same time step. The following code describes the adjusted transformation algorithm for non-parallel activities:

Algorithm 1

Pseudocode for selection of only one action with highest value

```
max = 0.0
max_position = -1
for ( i = n_N + 1, ..., n_N +n_A){
    if (v_i^t > a_c AND v_i^t > max){
            max = v_i^t
            if (max_position ≠-1)
                    v^t_max_position = 0.0
            max_position = i
            v_i^t = 1.0
    }else
            v_i^t = 0.0
}
```

When it is necessary to disable parallel activities, the algorithm finds the *Activity* node with the highest value after all nodes are calculated. If the found value exceeds a critical level a_c, then the activity is performed; otherwise, the agent does nothing

(alternatively returns to its default state or starts a free time activity). Within this single cycle, the algorithm finds the most necessary activity and rounds the values to zero or one.

4.4 Primary state, fitness and constant increments

The evaluation of a system is a critical topic in any field: the development and progress of any system always depend on some kind of feedback. Since the *Needs* in the FCM-NAS serve as indicators of success in particular tasks, their values can be easily converted into a general evaluation measure of the success (fitness) of an FCM.

In order to integrate the fitness into an FCM-NAS, a *Primary State* (*PS*) node is added to the *States*. Only *Needs* that are included in the evaluation of the system can affect the *PS* node. If the recommended convention for the setting of the *Needs* is met (a value of zero indicates maximal satisfaction; no action is required), then all these relations are negative. Therefore, the higher the value of a *Need* node, the lower the fitness of an FCM-NAS. Since fitness ranges from zero to one and is likely to change during the previous computation, its base value has to be reset to one before computations. The relation of the *PS* node to itself w_{pp} is also equal to one, where p is the position of this node in an FCM-NAS.

For example, Table 3 shows a part of the matrix R of the FCM-NAS in an artificial life model. This part contains three *Needs* and the *Primary State* node. The last row in the table contains the relations of all nodes to the *PS* node. Since the value of the *Needs* decreases with increasing satisfaction, these relations are negative, and the *PS* node is set to 1.0 before computation begins. In this example, *Hunger* and *Thirst* are more important for success than *Reproductive Need*; they therefore have a much higher negative impact on the fitness of an individual.

If the *PS* node starts at 1.0, then it can serve as a constant increment to any node. For the *PS* node c_p and a node c_i, the relation w_{pi} guarantees that a steady increase (or decrease) is added to node c_i at every time step. This relationship is especially useful for the stable growth of *Needs*. The last column of Table 3 shows the positive relations of the *PS* node to the *Needs*. These relations simulate a constant increase of the needs over time. For instance, *Thirst* would be increased by 0.1 per every time step (with default frequency parameter $g = 1.0$).

Table 3 An example of matrix R of an FCM with fitness

	Hunger	Thirst	Reproductive need	Primary state	…
Hunger	0	0	0	0.05	
Thirst	0	0	0	0.1	
Reproductive need	0	0	0	0.01	
Primary state	−0.5	−0.5	−0.2	1	
…					…

4.5 Necessity

The necessity of actions provides the FCM-NAS with another useful measure for decision-making. If several *Needs* have high values, then the decision-making prefers actions that relate to the most vital need. This feature is useful for the decision-making process in an FCM-NAS with disabled parallel activities. For example, in the artificial life model, an individual with values of both *Hunger* and *Reproductive Need* of 1.0 will prefer activities that lead to the meeting of a more critical need. In the case shown in Table 3, the individual would select feeding rather than reproduction, since the *Hunger* need affects the *PS* more than the *Reproductive Need* does.

This feature uses states to evaluate the necessity of the *Needs*. The *PS* node is recommended since at least one fitness value is required. More states can represent the different fitness functions of the system, and the necessity feature covers even this possibility. In the first step, a designer identifies those *States* that are used as fitness values and compares their necessity to the system. The constant d_i represents these evaluations for all *States* in the form of values ranging from zero to one, where $d_i = 1.0$ means that *State* node c_i has the highest priority and vice versa. The necessity value for a *PS* node would therefore be at 1.0, other *States* would range from zero to one depending on their relative importance to the *PS* node, and all *States* nodes that do not serve as a fitness value (e.g., an external inputs) would be at 0.0. Note that index i does not start at one for d_i values since i represents the position of the node in an FCM-NAS that also contains other types of nodes that come before *States*.

The necessity of each *Need* node is calculated during initialization of the system according to their influence on the *States* and the corresponding values d_i:

$$e'_i = \sum_{\mathrm{j}=(n_N+n_A+1)}^{n} \left(-\mathrm{w_{ij}} \cdot \mathrm{d}_j\right) \quad for\, i = \{0, \ldots, n_N\} \tag{19}$$

Then the coefficients are adjusted by an increment k_n that shifts their values, so their mean is 1.0. In this way, setting absolute values of necessity does not affect the activation of *Activities*:

$$k_n = 1 - \frac{\sum_{\mathrm{i}=1}^{n_N} \left(e'_i\right)}{n_N} \tag{20}$$

$$e_i = k_n + e'_i \quad for\, i = \{0, \ldots, n_N\} \tag{21}$$

Then, e_i is the final necessity coefficient of *Need* node c_i and e'_i is the necessity value e_i without the compensation of the offset k_n. In the computation of an FCM-NAS, the necessity is used to calculate the *Activities* from the *Needs*. Equation (22) shows the placement of the necessity coefficient. Calculations of the effects of *Needs* to *Activities* have to be separated; otherwise the rest stays the same as in Eq. (13).

$$v_j^t = f_a\left[\left(\sum_{i=1}^{n_N} \boldsymbol{e_i} \cdot w_{ij} \cdot v_i^{t-1}\right) + \left(\sum_{i=(n_N+1)}^{n} w_{ij} \cdot v_i^{t-1}\right)\right] \quad \text{for } j = \{n_N + 1, \ldots, n_N + n_A\} \tag{22}$$

4.6 Summary of the solution

The process of computing the FCM-NAS occurs periodically in intervals with constant length. This procedure contains calculations of each class, reading values for input nodes, and the selection of activity. The fact that all classes are processed individually means it can be rearranged and also use new values of previously processed nodes. The algorithm of the full process is described by pseudocode in the appendix of this paper. Figure 3 shows the best possible order of computations which allows the decision-making part of the FCM-NAS to react to the current situation instead of the situation at the previous time step (more in Sect. 4.2). In the first phase, *Needs* are updated; then, computation of *States* can optionally use these new values. After this part, external inputs are imported to dedicated *States*. Alternatively, this import could appear at the very beginning since the calculation of nodes dedicated to external inputs would be ideally omitted. If these calculations are processed, however, they would overwrite the value after its import.

At this stage, the new updated value of *Primary State* is copied to an external variable, if this node is used for constant increments. In that case, the *PS* node would be reset to 1.0 before the computation of *Activities*. *Activities* are calculated during this stage; optionally using both updated sets of nodes to make the reactions of the decision-making immediate. Finally, the values of *Activities* go through the transformation function to select the activity (or activities) to perform. Lower-level decision-making or actuators then take control until the next computation.

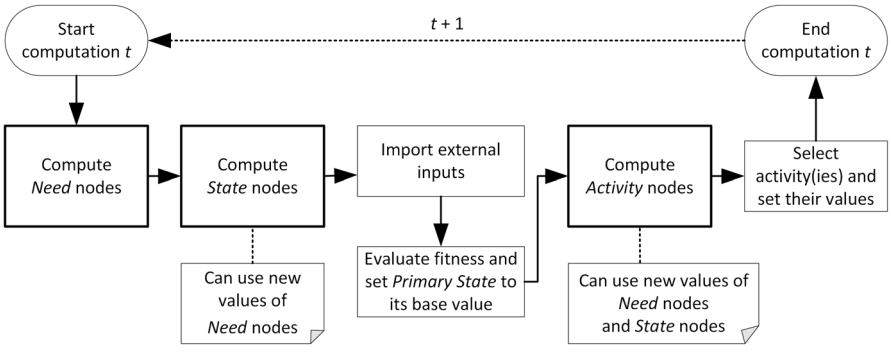

Fig. 3 Summary of the computation of the FCM-NAS optimized for autonomous agents with described features

5 Applications of FCM-NAS

This section demonstrates the proposed approach in two models: (1) an artificial life model aiming for evolution and testing artificial intelligence of autonomous agents, and (2) an ambient intelligence model, where the FCM-NAS processes the decision-making of simulated inhabitant of a smart environment.

5.1 FCM-NAS in an artificial life model

The proposed solution was implemented into the artificial life model *ALModel*, featuring a randomly generated two-dimensional environment with resources and thousands of individuals (see Fig. 4). The model was built on the NetLogo 5 platform, and is available for download at [32]. The simulation runs in discrete time steps (ticks), and uses an FCM-NAS as an artificial intelligence method. Each individual makes a decision about their activity at the beginning of every tick. Parallel activities are disabled, meaning that individuals cannot perform more than one activity per time step.

The proposed approach manages agents' behavior. A single FCM-NAS simulates needs, processes information from sensors, and performs decision-making. Each aspect of artificial life usually requires more than one node. For instance, the food management of individuals uses one node for the level of need (*Hunger*), two nodes for activities (*Feeding*, *Searching for Food*) and a *State* node as external input, which allows currently available food supplies to be taken into consideration. More concept nodes in any FCM usually mean more possibilities, but also that the FCM becomes more demanding in terms of performance.

Only the core areas of artificial life were implemented in the model. The behavior of individuals involves food, water, fatigue, danger, and reproduction. Artificial life requires evolution, and one of its elements is selection, which emerges through

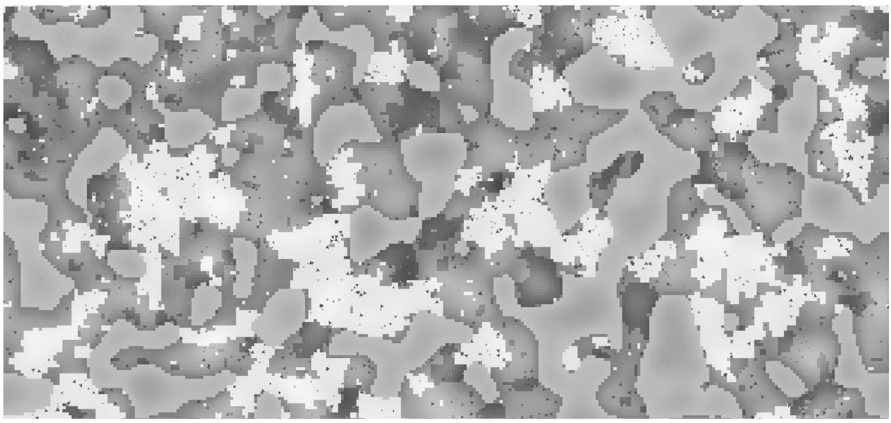

Fig. 4 Screenshot of the environment of the artificial life model [3]

Table 4 Classification of concept nodes in the *ALModel*

Needs	Activities	States
Hunger (c_1)	Feed (c_5)	Primary state (c_{13})
Thirst (c_2)	Drink (c_6)	Lack of food (c_{14})
Fatigue (c_3)	Sleep (c_7)	Lack of water (c_{15})
Reproductive N. (c_4)	Reproduce (c_8)	Lack of partners (c_{16})
	Search for food (c_9)	Danger (c_{17})
	Search for water (c_{10})	
	Search for partner (c_{11})	
	Escape (c_{12})	

competition for resources. In this model, food and water are these vital resources. Fatigue is closely connected with the availability of resources, and its level affects the general performance of the individual. Reproduction is the next essential part of evolution. Finally, the *State* node *Danger* with the activity *Escape* enables individuals to escape from predators; this is the only defense for most of the individuals.

Table 4 contains all concept nodes distributed into classes. *Needs* are set in the recommended way, meaning that a value of one represents the highest need, and zero means maximal satisfaction. *States* that represent the availability of a resource or partner are inverted to the lack of these entities, so a negative relationship is used instead of a positive one; in other words, the "*Food availability* causes *Feeding*" relationship is replaced by "*Lack of food* prevents *Feeding*," which is a more intuitive representation.

Figure 5 depicts an individual in the model; white arrows represent a flow of information, and dark arrows express a direction of influence or method call. The FCM-NAS reads data from sensors and inner states, and then makes a decision according to the current situation and sends the request to actuators. It manages behavior at a higher level, and the actuators try to fulfill the command within a given environment. The actuators perform the selected action while taking into

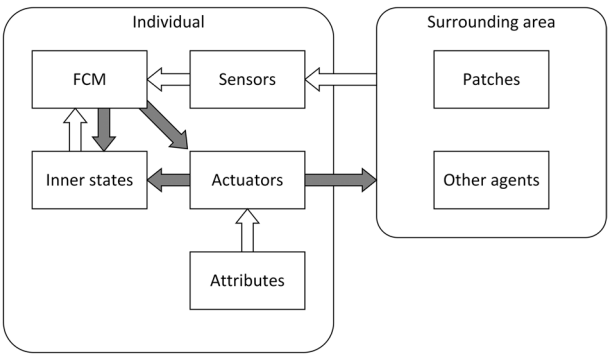

Fig. 5 Diagram of an individual in the *ALModel* [33]

consideration the individual's vicinity and attributes; for example, the FCM-NAS decides to search for food, and the actuators then direct the individual to the closest food source, if this is within sight, or alternatively to the location where the individual fed last time. If there is an obstacle (e.g., water) in the way, the actuators need to perform more difficult path-finding than just heading directly to the target location.

Table 5 shows the initial settings of matrix R. The top section (i.e., four rows of *Needs*) contains all relations to the *Need* nodes. This section represents a set of rules that determine the increase in the needs and the effects of activities in the model. These values are affected by the frequency setting of a model run.

The middle section (i.e., eight rows of *Activities*) represents the decision-making part of the FCM-NAS. In the *ALModel*, this section changes dynamically during a model run, through evolution. Its initial setting is therefore not essential, because the responsibility for correct behavior moves from the designer to the evolution of the model. Nevertheless, due to the dynamic environment, the optimal behavior at the beginning usually differs from the optimal behavior at later stages of a model run. Various species also often require different behavior patterns that better suit their attributes and types of diet.

Finally, the bottom section (i.e., five rows of *States*) contains mostly zeros, since four of these *States* are external inputs. The *PS* node begins computation at a value of 1.0, and is then lowered by *Needs*. The FCM-NAS uses this node for the fitness of individuals and the necessity feature. Since this is the only fitness node, its importance value d_1 is 1.0, while the values of the other *States* are 0.0.

Table 6 shows the middle section of the FCM-NAS after 89 generations. This particular matrix R represents a member of a successful small omnivore species. A few relationships did not prove to be useful and lost their influence on behavior, for example the negative relationship between *Hunger* and the *Reproduce* activity or the positive relationship of *Sleep* activity with itself. On the other hand, several new relations arose. For instance, this species learned over generations that taking a rest is beneficial immediately after escaping from a predator. Also, while searching for water, this species is less likely to flee if they spot a predator. Since predators often gather near water sources, thirsty individuals have to come closer to them than they would normally allow in any other situation.

Figure 6 shows 100 time steps of a single individual in the model. In this case, the frequency parameter was set to 0.6. The top plot in the figure depicts the development of the *Needs* and fitness. The bottom plot explains the changes in the values in the plot above with a log of the activities that this individual performed during the observed 100 time steps. During the first 25 time steps, the individual struggled to find resources: *Needs* were increasing, and its fitness was decreasing. Then this individual finally found resources, rested, and reproduced. Reproduction raised the values of its *Needs* due to the corresponding values set in matrix R (see Table 5, Column c_8). Except for *Sleep*, *Escape*, and *Searching*, all activities floor the value of the corresponding *Need* node in a single time step.

Individuals are capable of surviving over hundreds of thousands of time steps, increasing the population, and evolving. Verifying the solution on a larger scale of an entire population is problematic because the *PS* node enables only the evaluation of individuals. In the simulation, this fitness node is not essential for the

Table 5 Initial matrix of relations R in the *ALModel*

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}
Hunger	0	0	0	0	*−1*	0	*−0.005*	**0.3**	0	0	0	0	**0.01**	0	0	0	0
Thirst	0	0	0	0	**0.05**	*−1*	*−0.005*	**0.3**	0	0	0	0	**0.01**	0	0	0	0
Fatigue	**0.001**	**0.001**	0	0	0	0	*−0.2*	**0.3**	**0.005**	**0.005**	**0.005**	**0.1**	**0.005**	0	0	0	0
Reproductive N.	0	0	0	0	0	0	0	*−1*	0	0	0	0	**0.002**	0	0	0	0
Feed	**1**	0	0	0	0	0	0	0	0	0	0	0	0	*−0.5*	0	0	*−0.2*
Drink	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	*−0.5*	0	*−0.2*
Sleep	0	0	**0.8**	0	0	0	**0.2**	0	0	0	0	0	0	0	0	0	*−0.8*
Reproduce	*−0.2*	*−0.2*	*−0.2*	**1**	0	0	0	0	0	0	0	0	0	0	0	*−1*	*−0.8*
Search for food	**0.7**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Search for water	0	**0.7**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S. for partner	*−0.2*	*−0.2*	*−0.2*	**0.7**	0	0	0	0	0	0	0	0	0	0	0	0	0
Escape	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
Primary state	*−0.5*	*−0.55*	*−0.35*	*−0.1*	0	0	0	0	0	0	0	0	**1**	0	0	0	0
Lack of food	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Lack of water	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Lack of partners	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Danger	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6 The decision-making section of matrix *R* of an individual at a later stage of a model run (generation 89)

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}
Feed	**0.8**	−0.2	0.2	0.2	*−0.3*	0.0	−0.1	*−0.2*	0.0	0.1	−0.1	0.1	0.0	*−0.9*	**0.2**	−0.2	*−0.5*
Drink	−0.1	**1.0**	0.1	0.1	0.0	0.0	0.0	0.0	0.0	**0.4**	−0.2	−0.1	−0.1	0.1	*−0.4*	**0.2**	−0.2
Sleep	*−0.2*	0.1	**0.9**	0.0	0.1	**0.2**	−0.1	0.0	0.1	−0.2	0.1	**0.4**	0.1	0.2	0.1	0.0	*−0.8*
Reproduce	0.0	−0.2	*−0.5*	**1.0**	0.0	*−0.3*	0.0	−0.2	−0.1	**0.3**	0.2	−0.1	−0.2	−0.1	−0.2	*−0.8*	*−0.9*
S. for food	**0.3**	0.1	0.1	0.0	0.0	*−0.3*	**0.3**	0.1	−0.1	0.0	−0.1	0.0	*−0.2*	−0.1	0.0	−0.2	−0.1
S. for water	**0.2**	**0.8**	0.0	0.1	**0.2**	−0.1	0.1	0.2	**0.2**	0.0	0.0	*−0.2*	−0.1	−0.1	0.0	0.2	0.0
S. for partner	−0.1	*−0.5*	−0.1	**0.7**	0.2	0.2	0.2	−0.1	−0.2	−0.1	0.0	*−0.3*	0.0	**0.2**	0.0	*−0.2*	**0.3**
Escape	**0.2**	0.0	0.0	*−0.2*	−0.1	0.1	−0.1	0.0	0.1	*−0.4*	0.0	0.0	−0.1	−0.1	0.2	0.2	**0.9**

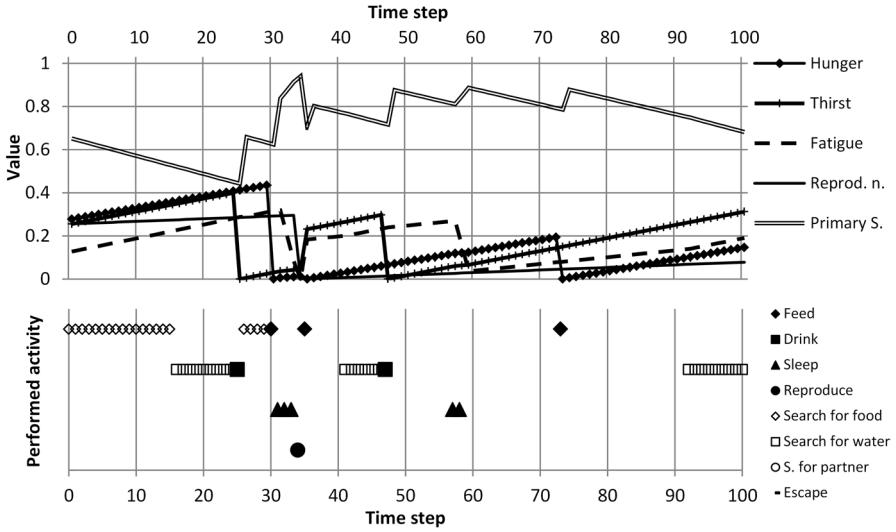

Fig. 6 *Need* and *Activity* nodes of an individual in the *ALModel* over 100 time steps

survivability of the entire species. Due to natural selection and the aggressive competition in the dynamic environment, successful species tend to prefer the survivability of the species as a whole over individual satisfaction; however, the *ALModel* offers various other measures.

The ratio of deaths caused by a lack of resources to total deaths offers one possibility for verifying the adaptation of the FCM-NAS in the model. Obviously, this ratio depends on the availability of resources in the dynamic environment; however, Fig. 7 shows the development of the ratio during a model run with a stable environment. To cancel out any other influences except adaptation, this environment maintained resources at stable levels and contained a constant number of individuals from only one species. The decrease in this ratio proves that the artificial intelligence of the individuals is capable of adaptation, even at the population scale.

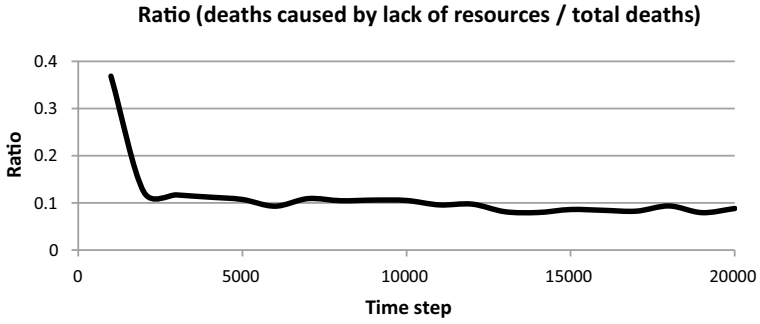

Fig. 7 Development of the ratio of deaths caused by lack of resources to total deaths

With evolutionary adaptation, the genomes of individuals (i.e., their attributes and behavior) develop during a model run. When the genome of an offspring is sufficiently differentiated from the original genome of its species, it founds a new species, and this process generates a phylogenetic tree. Figure 8 shows a phylogenetic tree of species produced in a single model run, in which each bubble represents a different species. The arrows point to the species that evolved from the species on the other end of the arrow, and sizes of the bubbles represent the numbers of individuals that were classified as members of that species at the moment of creating the graph (i.e., small crosses correspond to extinct species). The brightness of the bubbles depicts specific attributes of a species; in this case, it is the type of diet (i.e., a black bubble means a carnivore species; a brighter one a herbivore species). The experimental results presented here demonstrate that this decision-making method allows for rational behavior with multiple objectives, learning, and interactions.

The *ALModel* does not attempt to copy the real ecosystem precisely and offers generic individuals rather than specific animals or even biological classes. Despite that, evolution in the model successfully generates realistic patterns seen in real nature. The model has shown similar species abundance patterns that were observed in biological systems. Similarly to *EcoSim* in [34], several random samples were collected during a simulation run to provide data for comparisons with expected data calculated with Fisher's log series. Figure 9 shows species abundance distribution in communities generated in the exact model run that created the phylogenetic tree shown in Fig. 8. A community is a randomly selected sample of individuals in the model. Members of each species are counted, forming the distribution based on the number of individuals per species. This comparison shows a similar pattern in diversity and abundance of species created by real biological evolution as species generated in the *ALModel*.

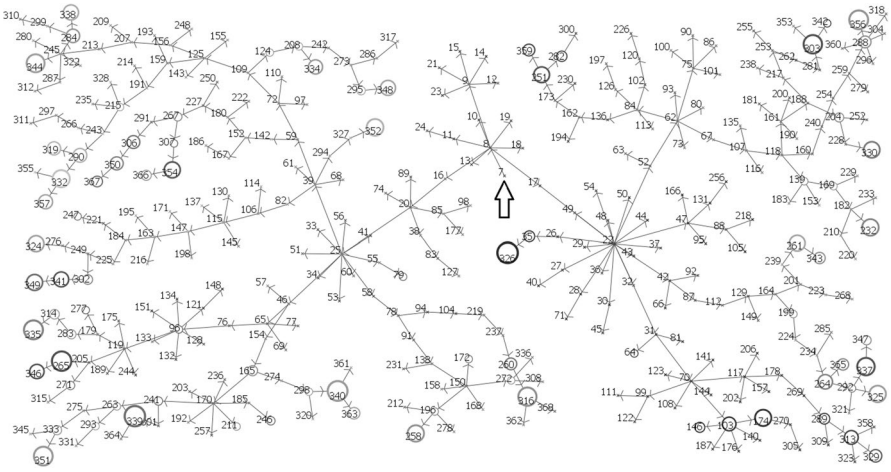

Fig. 8 A phylogenetic tree of species generated in a single run of the artificial life model (from the initial species 7 marked with an arrow; after over 297,000 time steps and 241 generations on average)

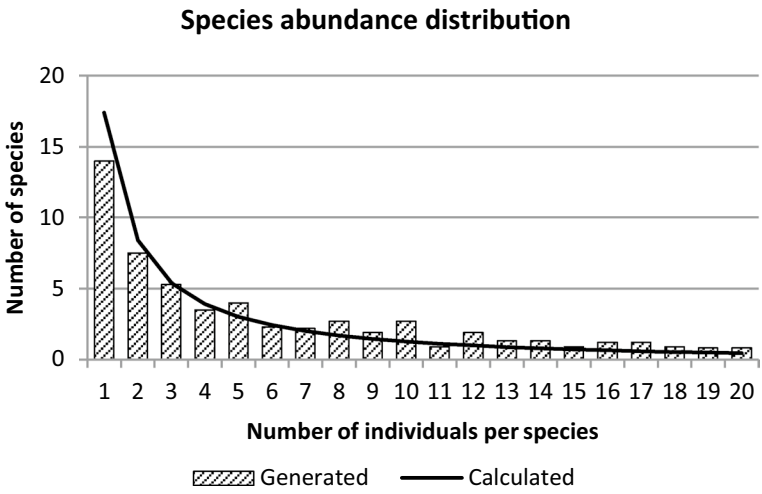

Fig. 9 Species abundance distribution of 10 samples (the number of individuals per sample is 500; the number of different species per sample is 61 on average) compared to Fisher's logseries [34] with parameters approximated from sample size ($\alpha = 18$; $x = 0.965$)

5.2 FCM-NAS in an ambient intelligence model

The proposed method was also implemented into an ambient intelligence model. The model was created during project GAČR "DEPIES - Decision Processes in Intelligent Environments" (n. 1511724S) on the AnyLogic platform (see Fig. 10). The finished environment and a basic smart home system required the decision-making of agents (virtual inhabitants) to allow running simulations in the model. Our objective was to develop a new approach to a human simulation that would combine both

Fig. 10 Screenshot of the apartment in the model designed on the AnyLogic platform

scenario-based and need-oriented behaviors. To accomplish this objective, standard scenario management required for routines was combined with an FCM-NAS, which provided need-oriented behavior and high-level decision-making [35].

This model supports multi-agent and multi-scenario simulations, conditions and requirements of activities, and hierarchical structure of contexts, activities, and actions. On top of these general features, an FCM-NAS added possibilities like delaying or interrupting activities due to unexpected events, managing non-planned tasks, and adaptation of behavior through machine learning.

Unlike in the previous case, the time-frame of the ambient intelligence model has to be more precise and is set in weeks or months instead of centuries required to observe any effects of evolution. The time-frame significantly changes priorities and required detail of simulation, hence configuration of an FCM-NAS. The agent's behavior covers the following needs: hunger, thirst, fatigue, boredom, hygiene need, toilet need, and social need. Each of these has a corresponding *Activity* node, which triggers one of the activities that are supposed to satisfy the need. The choice of the specific action to handle the need is the responsibility of lower-level decision-making.

All *States* except *Primary State* are external inputs. In this context, it means outside of the FCM-NAS, not necessarily external for the agent as a whole entity. Truth values of nodes *Planned*, *Delay*, and *NearPlanned* represent states of activity buffer in the schedule management of a scenario. These nodes help the FCM-NAS to decide whether there are essential planned activities to perform or it is free to cover needs. Values of states *Nighttime* and *Daytime* are global; hence their values are the same for all agents in a model run. These values allow agents to include time in decision-making. Other more precise variants would also be possible. For instance, nodes *Morning*, *Afternoon*, *Evening*, and *Night* would potentially allow a machine-learning algorithm to adjust behavior to day time better. With more complex scenarios, nodes for distinguishing weekdays from the weekend would also be very helpful. For this model, the FCM-NAS was designed as a proof of concept with the least nodes possible to allow fast testing and learning. Otherwise, additional *State* nodes for external inputs have relatively small performance impact because their computation is omitted. Table 7 lists all nodes in the designed FCM-NAS.

Figure 11 shows the structure of the agent Person. The white arrows represent a flow of information; the dark arrows express a direction of influence or control. The dashed arrow depicts the import of the scenario into the schedule. To decide, the FCM gathers data from the schedule and inner states. The schedule returns values representing if there is an activity that should be in progress right now or is about to start soon. Inner states provide values of all needs and the activity performed last time. FCM is able to make higher-level decisions, what the agent should do. It selects a general group of activities (a context) and forwards it to actuators, which pick specific action regarding the current location or conditions. These lower-level decisions are made by simple rules which select a specific action and ensure the right position of the agent. For example, when the agent is supposed to eat, the algorithm checks the agent's level of the corresponding need *Hunger*. If it is high, then the agent starts to cook a meal. Otherwise, it takes

Table 7 Classification of concept nodes in the ambient intelligence model [35]

Needs	Activities	States
Hunger (c_1)	HungerActivity (c_8)	PrimaryState (c_{16})
Thirst (c_2)	ThirstActivity (c_9)	Planned (c_{17})
Fatigue (c_3)	FatigueActivity (c_{10})	Delay (c_{18})
Boredom (c_4)	BoredomActivity (c_{11})	NearPlanned (c_{19})
HygieneNeed (c_5)	HygieneActivity (c_{12})	Nighttime (c_{20})
ToiletNeed (c_6)	ToiletActivity (c_{13})	Daytime (c_{21})
SocialNeed (c_7)	SocialActivity (c_{14})	
	PlannedActivity (c_{15})	

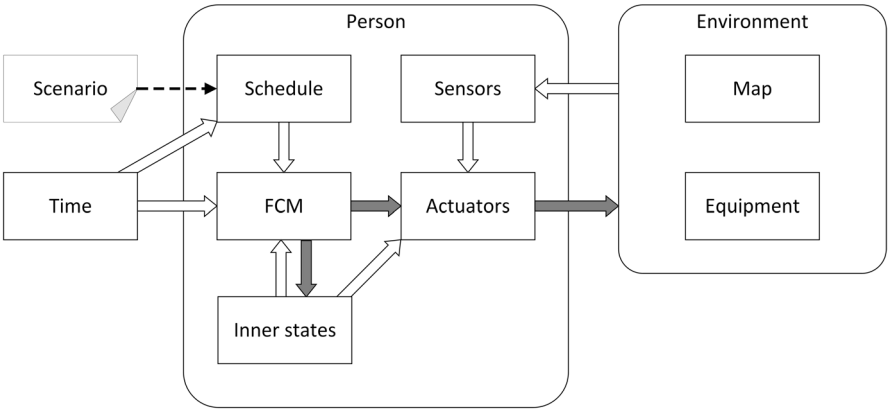

Fig. 11 Diagram of the agent Person; white arrows mean direction of information flow, dark arrows represent the direction of control or influence [35]

a snack. If the agent is not in a kitchen, it starts moving to the corresponding spot in the environment.

The model triggers the decision-making of each agent every minute. This interval can be arbitrarily tuned thanks to the adjustable frequency of computations in FCM-NAS. This process starts with higher-level decision-making using FCM-NAS and schedule management. As seen in Fig. 12, regardless of the FCM-NAS decision, the schedule always updates its activity buffer. Firstly, it checks all activities in the buffer for missed activities. There are two possible ways for activities to be missed: (1) the activity has not yet started and missed delay tolerance; (2) the activity started but has been interrupted and missed its latest finish time. After discards, the schedule adds the upcoming activities that either start in one hour or have a longer start tolerance interval. Then the schedule management sorts activities by their current value of priority function, so if the FCM-NAS decides to process a planned activity, then the first one from the buffer is performed.

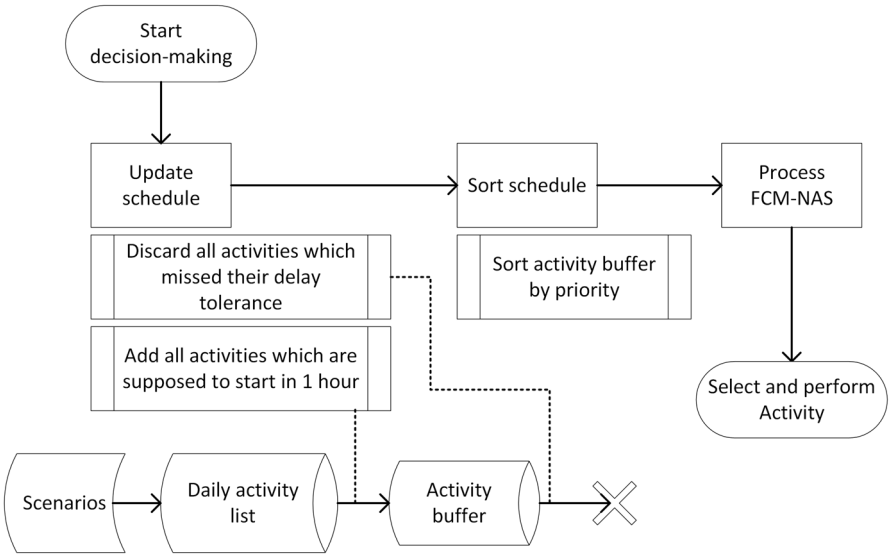

Fig. 12 Diagram of the higher-level decision-making process [35]

As in the previous model, genetic algorithms were used to configure and tune the relations of nodes. However, since this model does not include natural evolution, it needs to be implemented separately from the main simulation. This approach requires many repetitions of the same simulation with different configurations of decision-making. One of the ways to implement this is running parallel simulations corresponding with the size of the population. This model launches the whole population in the environment for a set number of days. To avoid any interactions between agents and their perceived environment, agents need to record any possible changes to the environment to local variables, and the ambient intelligence system must be turned off during this learning process. These measures ensure the same conditions for each agent and guarantee that only the behavior alone affects its final score.

After a given simulation period, fitness function evaluates each genotype, meaning the configuration of decision-making in this case. The fitness function in the model does not only rate behavior, but in the case of genetic algorithms, the fitness defines the direction of evolution that forms the intended goal of the learning process. As the decision-making method covers two different concepts, its fitness function requires at least two components. The first fitness component evaluates the need-oriented part of behavior based on the average success rate of satisfying needs during the simulation. It is possible through the *Primary State* node. The second component focuses on routine behavior and planned activities. It reflects how many of the planned activities were finished during the simulation while considering the importance of activities. The early experiments revealed the need for another component representing efficiency. Without it, the learning process led to the optimization of behavior corresponding to both decision-making parts; however, it still did

Table 8 Comparison of fitness values and time distributions of initial behavior with and without the third fitness component; results from an experiment with one specific scenario of a guest of a spa resort [35]

Set of activities (weight)	Initial (designed by hand)		Learned (2 fitness components)		Learned (3 fitness components)	
	Time distribution (%)	Fitness	Time distribution (%)	Fitness	Time distribution (%)	Fitness
Need-oriented (0.25)	13	0.978	58	0.974	8	0.963
Planned (0.5)	29	0.800	31	0.993	30	0.985
Free time (0.25)	54	0.540	4	0.040*	59	0.595
Walking	4%	–	7	–	3	–
Total fitness		0.780		0.750*		0.822

*Fitness component considering free time was not used for learning; it has been added in the table for comparison

not produce effective and reasonable behavior. Time distribution in Table 8 shows that an agent tends to spend any free time between planned activities to keep needs satisfied. This behavior seems to be always alerted and repeatedly triggers activities even before there is the actual need. That is the reason why the third component of the fitness function was introduced. All three components of fitness have a weight, which allows merging it into the final evaluation.

Experiments were conducted to examine the generated behavior of the agent Person in a scenario of a guest of a spa resort. The combination of routine scenario-based and need-oriented approaches, which is exclusive to this solution [35], proved it could autonomously manage dynamic agent's needs within the planned scenario by delaying less critical activities or omitting redundant ones. All the agent's activities with corresponding starting times were recorded during the day. The agent was performing planned activities and satisfying needs for a few simulated weeks in the model of ambient intelligence. Since the agent starts days with different inner

Table 9 Recording of the actual behavior of the agent in one morning and its comparison with the scenario [35]

Time	Activity	Scenario comparison
5:47	toileting	13 min early
5:58	medicine	17 min early
6:03	take_snack	Not planned
6:05	eat_snack	Not planned
6:11	watch_tv	Not planned
6:23	exercise	7 min early
6:39	medicine	6 min early
6:44	get_drink	Not planned
6:47	drink	Not planned
6:49	watch_tv	Not planned
7:00	eat_breakfast	Not hungry (skipped)
7:00	read_news	10 min early

values, the final order and starting times of activities may differ each day, although the scenario was always the same. Table 9 shows the morning of one of the simulated days in comparison with the original scenario.

6 Discussion and future work

The proposed approach offers a great variety of possible uses and modifications for adjusting to a specific environment. The basis of this approach was developed for an ambient intelligence system in the limited virtual environment of Second Life (unfortunately, this has not yet been published in English, and the thesis [36] is the only source). It was later extended with evolutionary principles, allowing individuals to learn over generations [3, 33]. The modularity of the FCM-NAS method allows for the modification that replaces the decision-making part of FCM-NAS with analytic hierarchy process [3]; this modification is focused on the computational speed of processing, and while it is significantly faster, it slightly lowers the complexity, leading to fewer learning possibilities in the method. This may be advantageous for some applications, especially in large scale models with thousands of individuals. Due to its modularity, other modifications may be possible for specific objectives such as a network of FCMs, dynamic matrix sizes, and so on. Also, the provided set of optional extensions for FCM-NAS is not an exclusive list as more specific adjustments could be added in future.

Recently, this method has been implemented for the simulation of human activity [35], which can aid in experiments with smart systems of ambient intelligence. For this purpose, it was connected to a schedule planner to simulate human routines and planning better. The result was cooperation between the dynamic schedule, which covers all planned activities, and the FCM-NAS, which handles needs, unplanned activities, and emergencies. The development of this model is still in progress.

For more complex systems, the transformation of a large FCM into several smaller ones seems to be an appropriate way of modeling complex structures with a low density of relationships. A group of FCMs structured as a hierarchy, or a network recomputes all concept nodes much more quickly than one large FCM with the same number of nodes. Different levels of control can also be separated, with clear responsibilities. In combination with the FCM-NAS modification, this distribution undoubtedly has great potential in the modeling of autonomous agents.

As stated at the beginning of this paper, this approach aims to provide a framework for decision-making in dynamic environments. As such, it requires a time component and an outside dynamic environment. The proposed approach is not appropriate for uses in static closed systems often used for predictions. Unlike these models, an FCM-NAS can act within simulations, complex models, or real-world systems, but it is not a self-sufficient model. Table 10 summarizes the advantages and disadvantages of the proposed method.

Table 10 Summary of advantages and disadvantages of the proposed approach over the original FCMs

Advantages	Disadvantages
Modularity	Requirement of a time component
Adjustability	Requirement of an outside dynamic environment
Systematic development	Additional steps during its development
Easy to implement optional extensions	
Clear human-readable structure for decision-making	

7 Conclusion

This paper proposes a new approach to the design of FCMs for the decision-making of autonomous systems. The FCM-NAS modification distributes concepts into three classes according to their purpose within the map, and these classes differ in their calculation and interpretation. This design offers several advantages and features that are very useful for various systems dealing with a dynamic environment.

This paper provides the mathematical background of the proposed design and gives several examples for clarity. In addition to the core of the FCM-NAS, it presents many optional attachments that provide additional help in designing autonomous agents. The paper also presents the complete design of the FCM-NAS for the artificial life model, together with testing of this modification in the final model. This testing verified the solution and all of its features, proving both its functionality and evolutionary adaptation. Although this adaptation is not a part of the proposed modification, FCMs usually require a certain learning or training process. This paper verifies that the FCM-NAS supports evolutionary algorithms. Moreover, due to its clear classification and modularity, any combination with other algorithms or extensions is even easier to implement than with the original general FCM.

The current development of the modification focuses on its combination with other methods that can provide additional features or improve existing ones. There are already modifications of this method; a combination with an analytic hierarchy process improves the performance of a model with thousands of agents, and with the addition of schedule management, FCM-NAS can believably simulate human activities, including both routine and unplanned, spontaneous activities.

In terms of its complexity and difficulty of development, FCM-NAS is complex, powerful and has a high learning ability while retaining human readability. Moreover, its modularity offers great potential for a wide range of uses in the modeling of autonomous agents.

Acknowledgments Support from the Excellence Project "Decision Support Systems: Principles and Applications 2" in the Faculty of Informatics and Management, University of Hradec Králové, is gratefully acknowledged.

Appendix

The following pseudocode includes the functions of the computation of FCM-NAS needed to process values every time step:

Algorithm 2

High-level procedure

```
processFCM(values, matrix){
    previousValues = copy(values)
    processNeeds()
    processStates()
    updateInputStateNodes()          //if there are any external input nodes
    processActivities()
    transformationOfActivities()     //select activity to perform
}
```

Algorithm 3

Computations of FCM-NAS sections

```
processNeeds(){
    for ( j = 1, ..., numberOfNeeds ){
            increment = 0.0
            for ( i = 1, ..., numberOfNodes ){
                    increment += matrix[i, j] · previousValues[i]
            }
    values[j] = transformation(previousValues[j] + (frequencyParameter · increment))
    }
}

processStates(){
    for (j = numberOfNeeds + numberOfActivities + 1, ..., firstInput){
            value = 0.0
            for (i = 1, ..., numberOfNodes){
                    value += matrix[i, j] · previousValues[i]
            }
            values[j] = transformation(value)
    }
    fitness = values[primaryState]
    values[primaryState] = 1.0
}

processActivities(){            //necessity is enabled in this version
    for (j = numberOfNeeds + 1, ..., numberOfNeeds + numberOfActivities){
            value = 0.0
            for (i = 1, ..., numberOfNeeds){
                    value += matrix[i, j] · previousValues[i] · necessityValues[i]
            }
            for (i = numberOfNeeds + 1, ..., numberOfNodes){
                    value += matrix[i, j] · previousValues[i]
            }
            values[j] = value
    }
}
```

Algorithm 4

Transformation with disabled parallel activities

```
transformationOfActivities(){    //
    max = 0.0
    maxPosition = -1
    for (i = numberOfNeeds + 1, ..., numberOfNeeds + numberOfActivities){
        if (values[i] > criticalValue  AND values[i] > max){
            max = values[i]
            if (maxPosition != -1)
                values[maxPosition] = 0.0
            maxPosition = i
            values[i] = 1.0
        }else
            values[i] = 0.0
    }
}
```

Algorithm 5

General transformation function keeping values of concept nodes in the interval [0, 1]

```
transformation(value){
    if (value > 1.0)
        value = 1.0
    if (value < 0.0)
        value = 0.0
    return value
}
```

References

1. B. Kosko, Fuzzy cognitive maps. Int. J. Man Mach. Stud. **24**, 65–75 (1986)
2. S. Russell, P. Norvig, *The Artificial Intelligence* (Prentice Hall Press, Upper Saddle River, 2010)
3. T. Nachazel, Analytic hierarchy process in artificial life model based on fuzzy cognitive maps. J. Ambient Intell. Smart Environ. **10**, 127–141 (2018)
4. L.A. Zadeh, Fuzzy logic. Computer **21**, 83–93 (1988)
5. O. Motlagh, Z. Jamaludin, S.H. Tang, W. Khaksar, An agile FCM for real-time modeling of dynamic and real-life systems. Evolv. Syst. **6**, 153–165 (2015)
6. Y.-H. Hsieh, I.-H. Chen, S.-T. Yuan, FCM-based customer expectation-driven service dispatch system. Soft. Comput. **18**, 359–378 (2013)
7. V. Senniappan, J. Subramanian, E.I. Papageorgiou, S. Mohan, Application of fuzzy cognitive maps for crack categorization in columns of reinforced concrete structures. Neural Comput. Appl. **28**, 107–117 (2017)
8. S. Ahmadi, C.H. Yeh, E.I. Papageorgiou, R. Martin, An FCM-FAHP approach for managing readiness-relevant activities for ERP implementation. Comput. Ind. Eng. **88**, 501–517 (2015)
9. G. Kyriakarakos, K. Patlitzianas, M. Damasiotis, D. Papastefanakis, A fuzzy cognitive maps decision support system for renewables local planning. Renew. Sustain. Energy Rev. **39**, 209–222 (2014)
10. L.S. Jayashree, N. Palakkal, E.I. Papageorgiou, K. Papageorgiou, Application of fuzzy cognitive maps in precision agriculture: a case study on coconut yield management of southern India's Malabar region. Neural Comput. Appl. **26**, 1963–1978 (2015)
11. P.J. Giabbanelli, R. Crutzen, Creating groups with similar expected behavioural response in randomized controlled trials: a fuzzy cognitive map approach. BMC Med. Res. Methodol. **14**, 1–19 (2014)

12. H. Barón, R. Crespo, J. Pascual Espada, O. Martínez, Assessment of learning in environments interactive through fuzzy cognitive maps. Soft. Comput. **19**, 1037–1050 (2015)
13. M. Mendonça, L.V.R. de Arruda, F. Neves-Jr, Cooperative autonomous agents based on dynamical fuzzy cognitive maps, in *Fuzzy Cognitive Maps for Applied Sciences and Engineering: From Fundamentals to Extensions and Learning Algorithms*, ed. by I.E. Papageorgiou (Springer, Berlin, 2014), pp. 159–175
14. A.L. Laureano-Cruces, A. Rodriguez-Garcia, Design and implementation of an educational virtual pet using the OCC theory. J. Ambient Intell. Humaniz. Comput. **3**, 61–71 (2011)
15. G. Acampora, V. Loia, A. Vitiello, Distributing emotional services in ambient intelligence through cognitive agents. SOCA **5**, 17–35 (2011)
16. P. Szwed, P. Skrzynski, W. Chmiel, Risk assessment for a video surveillance system based on fuzzy cognitive maps. Multimed. Tools Appl. **75**, 10667–10690 (2016)
17. R.T. Jones, E. Connors, M. Mossey, J. Hyatt, N. Hansen, M. Endsley, Using fuzzy cognitive mapping techniques to model situation awareness for army infantry platoon leaders. Comput. Math. Organ. Theory **17**, 272–295 (2011)
18. R. Furfaro, W. Fink, J.S. Kargel, Autonomous real-time landing site selection for Venus and Titan using evolutionary fuzzy cognitive maps. Appl. Soft Comput. **12**, 3825–3839 (2012)
19. W. Stach, L. Kurgan, W. Pedrycz, A divide and conquer method for learning large fuzzy cognitive maps. Fuzzy Sets Syst. **161**, 2515–2532 (2010)
20. A. Jose, Dynamic fuzzy cognitive maps for the supervision of multiagent systems, in *Fuzzy Cognitive Maps: Advances in Theory, Methodologies, Tools and Applications*, ed. by M. Glykas (Springer, Berlin, 2010), pp. 307–324
21. M. Mendonça, L.V.R. de Arruda, I.R. Chrun, E.S. da Silva, Hybrid dynamic fuzzy cognitive maps evolution for autonomous navigation system, in *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)* (2015), pp. 1–7
22. M. Mendonça, E. Papageorgiou, L. Botoni de Souza, P. Soares, R. Barros, Dynamic fuzzy cognitive maps embedded and intelligent controllers applied inindustrial mixer process. Int. J. Adv. Syst. Meas. **10**, 222–233 (2017)
23. A. Mourhir, E. Papageorgiou, Empirical comparison of fuzzy cognitive maps and dynamic rule-based fuzzy cognitive maps. Presented at the ICAS 2017, Barcelona, Spain (2017)
24. K. Poczęta, Ł. Kubuś, A. Yastrebov, E.I. Papageorgiou, Learning fuzzy cognitive maps using evolutionary algorithm based on system performance indicators, in *International Conference Automation*, Cham (2017), pp. 554–564
25. J. Liu, Y. Chi, C. Zhu, A dynamic multiagent genetic algorithm for gene regulatory network reconstruction based on fuzzy cognitive maps. IEEE Trans. Fuzzy Syst. **24**, 419–431 (2016)
26. P. Giabbanelli, M. Fattoruso, M. L. Norman, CoFluences: simulating the spread of social influences via a hybrid agent-based/fuzzy cognitive maps architecture. Presented at the Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, Chicago, IL, USA (2019)
27. P.J. Giabbanelli, S.A. Gray, P. Aminpour, Combining fuzzy cognitive maps with agent-based modeling: frameworks and pitfalls of a powerful hybrid modeling approach to understand human-environment interactions. Environ. Model Softw. **95**, 320–325 (2017)
28. M. Stula, D. Krstinic, L. Seric, Intelligent forest fire monitoring system. Inf. Syst. Front. **14**, 725–739 (2011)
29. P. Leong, C. Miao, Fuzzy cognitive agents in shared virtual worlds, in *2005 International Conference on Cyberworlds (CW'05)* (2005)
30. R. Gras, A. Golestani, M. Hosseini, M. Khater, Y. Farahani, M. Mashayekhi et al., Ecosim: an individual-based platform for studying evolution, in *European Conference on Artificial Life* (2011), pp. 284–286
31. M. Khater, R. Gras, Adaptation and genomic evolution in EcoSim, in *From Animals to Animats 12: 12th International Conference on Simulation of Adaptive Behavior, SAB 2012, Odense, Denmark, August 27–30, 2012. Proceedings*, ed. by T. Ziemke, C. Balkenius, J. Hallam (Springer, Berlin, 2012), pp. 219–229
32. T. Nachazel, NetLogo User Community Models: ALModel (2016). http://ccl.northwestern.edu/netlogo/models/community/ALModel. Accessed 29 Mar 2020
33. T. Nachazel, Optimization of decision-making in artificial life model based on fuzzy cognitive maps, in *2015 International Conference on Intelligent Environments (IE)* (2015) pp. 136–139

34. D. Devaurs, R. Gras, Species abundance patterns in an ecosystem simulation studied through Fisher's logseries. Simul. Model. Pract. Theory **18**, 100–123 (2010)
35. T. Nachazel, Human Activities Simulation Based on Fuzzy Cognitive Maps, Ph.D. Doctoral thesis, Department of Information Technologies, University of Hradec Králové, Hradec Králové, 2020
36. T. Nacházel, Inteligentní systémy ve virtuálním prostředí, Bc. Bachelor thesis, Faculty of Informatics and Management, University of Hradec Králové, Hradec Králové, 2012