Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Calibrating the Performance Assessment Mechanism in Virtual Laboratories with Machine Learning: Using a Non-Hierarchical and a Hierarchical Variation

Vasilis Zafeiropoulos (vasiliszaf@eap.gr)
Hellenic Open University: Elleniko Anoikto Panepistemio https://orcid.org/0000-0003-0120-0488
Dimitris Kalles
Hellenic Open University: Elleniko Anoikto Panepistemio

Research Article

Keywords: Virtual Labs, Automated Assessment, Genetic Algorithms, Artificial Neural Networks

Posted Date: May 10th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1407461/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Calibrating the Performance Assessment Mechanism in Virtual Laboratories with Machine Learning: Using a Non-Hierarchical and a Hierarchical Variation

Vasilis Zafeiropoulos[1] (vasiliszaf@eap.gr)

Dimitris Kalles[1] (kalles@eap.gr)

1. Hellenic Open University, School of Science and Technology (Patras, Greece).

Abstract - Science universities are constantly confronted with the challenge of offering efficient lab training to their students while securing their safety and protecting the lab equipment from damages; hence the needs for the development of distance learnings tool for students to be trained virtually and safely in using the various lab instruments and performing experiments. Furthermore, as the students are evaluated for their laboratory performance at the on-site labs, the need for their assessment at the virtual labs is also present; hence the need for an incorporated assessment mechanism in the virtual lab in charge of the evaluation of the user's performance. For the assessment mechanism to be reliable though and devoid of the designer's bias, it may need calibration with machine learning. Hellenic Open University has developed its own virtual biology laboratory, Onlabs, which simulates its on-site one for its students to be trained and evaluated at. Considering the evaluation of the user's performance in Onlabs, it is made with respect to particular experiments and is based on an embedded scoring algorithm. The latter has also been optimized with the use of two separate machine learning methods, particularly a genetic algorithm and an artificial neural network. Moreover, within the context of the experimental procedure of microscoping, the scoring algorithm was redesigned in a hierarchical fashion in order to incorporate background knowledge about the procedure. In this paper, we present the two machine learning methods used for the optimization of the assessment mechanism in a non-hierarchical and a hierarchical variation and compare the results within a particular variation as well as within a particular method.

Keywords: Virtual Labs, Automated Assessment, Genetic Algorithms, Artificial Neural Networks

1. Introduction

One of the most delicate tasks that science universities are confronted with is the teaching and examination of their students in using the laboratory equipment and successfully carrying out experiments. Serious risks lurk when making use of an on-site laboratory, such as those of accidents and equipment damages, restricting therefore the full use of the various instruments as well as the possibility of learning by trial-and-error. Distance universities offering science education, particularly, face another problem, too, that of not having their students at a regular basis (or at all) at their laboratory facilities, making their lab training and evaluation even harder.

Those problems are abated to a great extent by the development of virtual laboratories as training platforms complementing to on-site lab training. In virtual labs, the user would make unlimited virtual use of the various instruments and perform experiments without the fear of equipment damages or accidents. They would also have the opportunity to have their performance been evaluated with respect to conducting a particular experiment either by a human expert supervising the session or by the virtual lab software itself.

In this fashion, Hellenic Open University (HOU), offering distance education, among others, in Natural Sciences, has been developing Onlabs, a 3D virtual biology laboratory for the lab training and evaluation of its undergraduate and postgraduate science students.

1.1. Hellenic Open University's Virtual Lab: Onlabs

Onlabs is an interactive virtual lab used for the distance laboratory education of undergraduate and post-graduate biology students of HOU. Unlike on-site lab training, in Onlabs, students can fearlessly experiment with the lab instruments and equipment in virtually unlimited ways, even making improper use of some of them, and learn by their own mistakes. They can even get guided with voice and text throughout a particular experiment or get their performance with respect to that experiment evaluated by the system and be given a numerical score for it.

Nevertheless, by no means does virtual experimentation in Onlabs substitute the on-site lab training; instead, it complements it. Students are being handed with its latest version every year and are expected to use it at home before they use the on-site laboratory.

Onlabs resembles a modern computer game; it contains state-of-the-art 3D graphics while the user interacts with it through the keyboard and the mouse and has tasks and missions to follow. Specifically, the user-environment interaction is similar to that of an adventure game; that is, they navigate with the arrow keys and they press buttons, turn knobs, combine specific objects etc. with the mouse.

Fig.1 shows a screenshot of Onlabs.

Fig.1 Virtual laboratory of Hellenic Open University (Onlabs)

1.1.1. Onlabs's Versions and Playing Modes

Between 2012-2016, Onlabs was developed with Hive3D[1] while from 2016 until today it's been being developed with Unity, both of them being high-end 3D game engines. Several versions of Onlabs have been released up to now, the latest stable one being version 2.1.2, released in November 2019. Currently, we have been developing version 3.0, a beta version of which was released in October 2020.

Onlabs's version 2.1.2, contains two distinct simulated experimental procedures and three different modes of playing. The procedures are those of *microscoping of a test specimen*, in which the user is supposed to set up the microscope, create a test specimen and microscope it with all objective lenses of the photonic microscope; and the *preparation of 500ml of 10X TBE water solution*, for which the user weights 17.4gr of Boric Acid and 54gr of Trizma Base powders on the electronic scale, dissolves them in water with the magnetic stirrer and adds extra water as well as EDTA pH 8.0 to the produced solution. The playing modes are those of *Instruction*, where the user is instructed by voice and text in completing the selected procedure, each time being allowed to perform only the suggested move; *Evaluation*, where the user can make any move they want within the selected procedure and receives an evaluation for their performance; and *Experimentation*, where the user may make any action they want (provided it is included in the game) and handles all the instruments and equipment from both procedures without being evaluated.

Onlabs beta version 3.0 contains two new experimental procedures which respectively supersede the procedures in version 2.1.2; *microscoping*, like in the previous versions, and *electrophoresis*, whose first part is the preparation of 500ml of 10X TBE water solution. But while microscoping is available in both modes of Instruction and Evaluation, electrophoresis is in neither of them;

[1] Released by Eyelead, a computer game company based in Athens, Greece.

however, electrophoresis instruments with their full functionality are included in the Experimentation Mode along with those of microscoping.

Onlabs 2.1.2 and Onlabs 3.0 beta can be downloaded and run from the project's website[2].

1.1.2. Onlabs's Embedded Assessment Mechanism

Onlabs assessment mechanism is set to revolve mainly around two factors; the actions (such as equipment handling) made for the completion of the experiment as well as the order in which those actions were made. As the various actions are, naturally, not of the same importance, different weights, concerning either the completion rate or the order of chosen actions, are assigned to each one of them.

Initially, such weights are defined based on suggestions by instructors and are, therefore, most probably, only partially accurate. Thus, Onlabs's assessment mechanism, which depends on those weights, is far from optimal. To adjust the assessment mechanism, i.e., to calibrate its weights, human-computer learning interaction is introduced in Onlabs, with a variety of machine learning techniques, to capture to some extent the grading/assessment decisions of biology instructors. Especially as regards the calibration of the weights concerning the completion rate of an experiment, we use either a Genetic Algorithm (GA) or an Artificial Neural Network (ANN), depending on the type of completion rate metric used. The selected method is based on a number of training examples consisting of various playing sessions by a human user and their evaluations by several different human experts. At the same time, for the weights concerning the order of the actions made in an experiment, a variant of Reinforcement Learning (RL) is used, based on various auto-playing sessions by a Non-Playable Character. This paper deals only with the machine learning techniques applied on the completion rate, i.e., the GA and the ANN.

The completion rate metric originally designed is "flat", meaning that the actions contributing to it are not grouped into sub-procedures. In this paper, apart from the original "flat" design of completion rate, we adopt a hierarchical one, in which the various actions are divided into subgroups according to the conceptual sub-procedures they belong to (e.g., testing the various microscope knobs belong conceptually to the sub-procedure of "knob testing") and each sub-procedure contributes as a whole to the completion rate. Then the respective machine learning techniques (the GA and the ANN) are applied on that hierarchical variation of scoring and the results are compared to the ones of the "flat" prototype.

The implementation of the machine learning training in Onlabs is developed in a separate version, Onlabs ML 1.0, which can be downloaded and tried from our project's website like the other versions of the software.

1.1.3. Evaluation of Onlabs

Onlabs's first version 0.1 was evaluated by undergraduate biology students at HOU with the System Usability Questionnaire (SUS) for systems engineering, developed by John Brooke of Digital Equipment Corporation in Great Britain, in 1996 (Brooke, 1996). The feedback received was encouraging; Onlabs was evaluated as easy to use while the students who used it improved their performance in the physical lab afterwards (Zafeiropoulos et al., 2014, 2016). Versions 0.2,

[2] http://onlabs.eap.gr/

0.2.1 and 0.2.3 were also evaluated, this time by other universities' undergraduate students as well as high school teachers in various disciplines of science, making use of the same, yet now adjusted, questionnaire. The evaluation results were analogous to the ones of version 0.1's evaluation.

For Onlabs's latest version 2.1, a more solid evaluation was done. Particularly, two educational methods with respect to the previously mentioned microscoping of a test specimen procedure were used and compared to each other, those of the routine on-site lab tutorial method and a proposed educational scenario of using Onlabs with a Skype session running at the same time. The learning results by various students were assessed with the use of Pre and Post-Tests. The Pre-Tests scores indicated that Onlabs experience provided the test subjects with higher baseline knowledge while the Post-Tests scores indicated that students who used Onlabs outperformed those who hadn't used it (Paxinou et al., 2017, 2018).

2. Background & Related Work

As mentioned in the introduction, a virtual biology laboratory could serve as a powerful tool for initiating students to the daily routine in the on-site laboratory, enhancing and further enriching their practical experience (virtually though) and offering them the opportunity to experiment safely and unrestrictedly on things they could not do in reality and learn by trial-and-error. Such realistic and instructive virtual labs are Labster, developed by the Danish multi-national company of the same name[3], and Learnexx 3D, developed by Solvexx Solutions Ltd, based in the UK[4].

Learning a subject by interacting through a virtual class is considered to be more efficient than attending a lecture on a subject or reading books on that subject (de Freitas, 2008). Furthermore, virtual classes provide shy or restrained students with the opportunity to perform learning actions, such as asking questions, that they would probably be reluctant to do in a real class (Maratou, 2012).

Another characteristic of Onlabs is its embedded assessment mechanism, which as mentioned in the introduction, is used for the evaluation of the user's performance with respect to a particular experiment. In computer games, the embedded mechanisms for the evaluation of the player's performance fall under the term 'scoring'.

Scoring has been present since the first arcade games in the history of personal computers. Since then, the classical 2D arcade computer games have evolved into state-of-the-art 3D multi-player ones, and along with them, scoring has diversified into more abstract measures like "experience points" and "skill points" ("Score (Game)," 2021). However, while computer games have also evolved into serious games and educational virtual worlds, the performance evaluation of the various users in them is mainly done externally by their tutors and not by hard-coded scoring algorithms (Bellotti et al., 2013). In fact, explicit evaluation mechanisms in serious games have been proposed as a major research direction (Bellotti et al., 2010).

For the purpose of developing reliable and robust automatic assessment methods, Mislevy et al. (2003, 2006) proposed Evidence-Centered Design (ECD). In ECD, special focus is laid on evidentiary

[3] https://www.labster.com/
[4] http://learnexx.com/

reasoning, i.e., data drawn from information and observations that facilitate inferences about unobservable aspects of an examinee's competence level in given performance situations. Being theoretically based on ECD, stealth assessment in games has been suggested (Shute, 2011; Shute & Ventura, 2013), which consists of a silent process by which performance data are being collected during playing and inferences are being made with Bayesian Networks about the player's skills. Such games that stealth assessment has been designed and incorporated into are Taiga Park, an immersive 3D role-playing game for middle school kids to improve their knowledge in ecology and their skills in scientific inquiry (Shute, 2011); Oblivion, a first person 3D role-playing game set in a medieval world, developed by Beshesda Softworks (Shute, 2011), and Newton's Playground, a computer game for students to be acquainted through 2D physics simulations with various physical quantities such gravity, mass, kinetic energy and transfer of momentum (Shute & Ventura, 2013).

Bayesian Networks have also been used for the assessment of university students' performance in a virtual electronic laboratory simulating in a realistic way various undergraduate electronic engineering curriculum-based laboratory activities at Portsmouth University (Achumba, 2011). In a similar fashion, a performance assessment mechanism with Artificial Neural Networks has been designed and incorporated into the IMMEX simulation at UCLA; the latter originally consisted of a simulated patient whose clinical immunology disorder the university's medical students were supposed to diagnose, but later its cognitive paradigm of problem solving was extended to other scientific fields as well (Stevens & Casillas, 2006).

Onlabs's design resembles the one of an adventure game; thus, a declarative approach in the scoring assessment is suitable to some extent. Nevertheless, as there are plenty of tools and instruments in the game whose internal states are in numerous cases defined arithmetically, a more quantitative approach has been adopted.

Resembling an adventure game, Onlabs also consists of a suitable test-bed for the training of artificial agents, as that particular category of computer games has been proposed for (Amir & Doyle, 2002; Hlubocky & Amir, 2004). Several machine learning applications have been developed within the framework of an adventure game. Such is the application of *Sophie's Kitchen*, developed by Thomaz & Breazeal (2008, 2006). In this, Sophie, a robot-like Non-Playable Character agent tries to prepare a cake by using the appropriate ingredients together and baking it in the oven; in each training session, the expert gives Sophie several feedbacks upon which Sophie learns how to properly make a cake with Reinforcement Learning.

In recent years, adventure games and particularly the text-based ones, have re-emerged as platforms for machine learning. In those, several techniques have been used for the training of NPCs, such as Deep Reinforcement Learning (Ammanabrolu & Riedl, 2019; Narasimhan et al., 2015) and Artificial Neural Networks (Kostka et al., 2017; Robitzski, 2019).

Despite ANNs not being popular for computer games because of their black-box nature (Robbins, 2019), there have been various studies and applications of them in gaming, for the classification of opponents (Charles et al., 2008), the exploration of arcade game levels (Luo, 2019) and the control of the fight or flight responses of NPCs (Robbins, 2019). In the last two decades, Genetic Algorithms have also been proposed for computer games, for the purpose of adjusting the NPCs' behavior (Arora, 2019; Charles et al., 2008; Martin, 2011; Mendonça et al., 2008).

3. Conceptual Modeling

Onlabs's virtual environment is discrete; that is, the objects in it are perceived as separate from one another and can be manipulated by an agent only through a finite set of actions. Moreover, the environment is deterministic; that is, when a specific action is performed, the produced new state of the environment is uniquely (in other words, non-stochastically) determined by the action and the environment's previous state (Russell & Norvig, 2003).

In Onlabs's virtual environment lie several different *entities*. Those entities can either be *characters* or "inanimate" *objects*. The main existing character in Onlabs is the *human agent*, namely the Ego Character, controlled by the user through the application's interface. In terms of both design and development, entities are handled by classes. A *class* is an abstract representation of one or more kindred entities; for example, all bottles in Onlabs (wash bottle, EDTA bottle) are *instances* of the *bottle* class. Furthermore, a class may be a specialization of a base class, or in programming terms, inherit from it; for example, the *bottle* class inherits from the *vessel* one, meaning that bottles share vessels' basic traits as well as having their own particular ones. For the sake of simplicity, we'll be referring to the various entities with their class names, unless we deal with two or more of a kind, so, in that case, we'll be using the names of the respective instances.

Most inanimate entities in Onlabs have specific actions that can be performed on them by the user. Such actions are *Pick Up* (collecting an object to the inventory), *Press* (valid for buttons, switches and triggers), *Rotate* (valid for knobs) and *Use With* (combining an object with an another one).

Each class has several *features*. Those can be qualitative (alphanumeric) or quantitative (numeric), depending on the *values* they can take; for example, the *state* feature of microscope's *AC switch* class is qualitative, taking the values of 'ON' and 'OFF', while the *position* feature of *aperture knob* class is quantitative, taking integer values from 0 to 40. The human user can alter those feature values directly, by performing various actions on the respective entities, or indirectly, by performing actions on other entities which get their features' values altered and those value changes cause in turn the values of the desired features of the entities on focus to change, too. For example, pressing the microscope's *AC switch* changes the latter's *state* feature from 'ON' to 'OFF' while using the microscope's *plug* with the *socket* changes the *microscope* entity's *connection status* feature from "disconnected" to "connected to socket" (Zafeiropoulos et al., 2014, 2016).

3.1. State-Transition Diagrams

When applicable, state-transition diagrams are used for the depiction of the value changes that take place in Onlabs (Yourdon, 1989). For the last two examples, the diagrams are shown in Fig.2 and Fig.3, respectively.

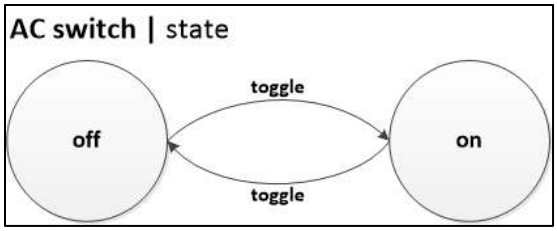

Fig.2 State-Transition Diagram of AC switch's state feature

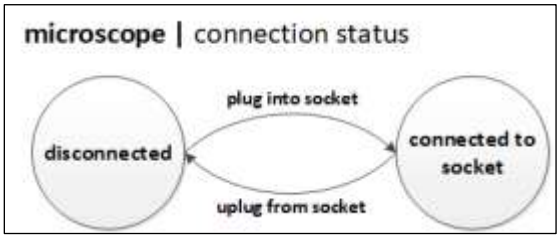

Fig.3 State-Transition Diagram of microscope's connection status feature

4. Assessment Mechanism

As mentioned in the introduction, Onlabs's scoring takes two metrics into consideration; the *success rate*, reflecting the steps that the user has made for the completion of a particular experiment and the *penalty points*, reflecting the order that those steps have been performed.

The necessary steps for the successful completion of the microscoping procedure are those listed in Table 1.

Table 1. Steps necessary for the successful completion of microscoping procedure.

1	Connect the microscope into the socket *microscope* | *connection* ← 'connected to socket'
2	Turn the microscope light on *AC switch* | *state* ← 'ON'
3	Set light intensity to 2 3rds of maximum *light intensity knob* | *position* ← ¾ · MaxPosition
4	Set iris fully open *aperture knob* | *position* ← MaxPosition
5	Lift the condenser lens to the top position *condenser* | *height* ← MaxHeight
6	Set lens 4X active *revolving nosepiece* | *active focus* ← 4
7	Test coarse focus knob

	coarse focus knob | *tested* ← 'yes'
8	Test fine focus knob *fine focus knob* | *tested* ← 'yes'
9	Test stage knob *stage knob* | *tested* ← 'yes'
10	Test specimen holder knob *test specimen knob* | *tested* ← 'yes'
11	Put test specimen on stage *stage* | *attached specimen* ← 'yes'
12	Enter microscope mode *Ego* | *mode* ← 'microscoping'
13	Focus with lens 4X *microscoping* | *clearness* [*revolving nosepiece* | *active focus* = 4] ← 1
14	Focus with lens 10X *microscoping* | *clearness* [*revolving nosepiece* | *active focus* = 10] ← 1
15	Focus with lens 40X *microscoping* | *clearness* [*revolving nosepiece* | *active focus* = 40] ← 1
16	Focus with lens 100X *microscoping* | *clearness* [*revolving nosepiece* | *active focus* = 100] ← 1

One sees from Table 1 which particular value assignment takes place at each step. Some of those values are alphanumeric while the majority of them are numeric. We start by calculating the *individual scores* for the various values; then we proceed into the overall scoring assessment.

For the calculation of the individual scores to be made, two separate processes must first take place: the *quantification* of the alphanumeric values and the *normalization* of the numeric ones.

4.1. Quantization

The first assignment in Table 1 consists of the *microscope*'s *connection status* feature taking the value of 'connected to socket'. Here are only two states, 'disconnected' and 'connected to socket'. The quantification to be done here is rather obvious: the former is converted to 0 and the latter to 1.

The same quantification is applied for the next value assignment at Step 2, as *AC switch*'s *state* feature also gets only two values, 'OFF' and 'ON', so 'OFF' becomes 0 and 'ON' becomes 1. Likewise, the quantification of the value of *stage*'s *attached specimen* feature is 0 when it's null and 1 when it points at a test specimen.

Similar are the quantifications for the alphanumeric values of other changeable features.

4.2. Normalization

All the afore-mentioned values, when quantified, were at the same time normalized; that is, they range in [0,1] (in fact, 0 and 1 are the only acceptable values within this interval). In other words, considering the features in question (*microscope*'s *connection status*, *AC switch*'s *state* and *stage*'s *attached specimen*), their quantified *initial* values are 0 while their *optimal* ones are 1.

We will now show how the normalization is possible in a case of a numerical value not ranging between 0 and 1, by focusing on the example of the *position* feature of the *aperture knob* of the microscope which configures the opening of the microscope's iris and the light getting through it. Its value ranges from 0 (the initial value, where the iris is closed) to 40 (the optimal value, where the iris is fully open). What we therefore need is a *normalization function* which would particularly return 0 when *position*'s value is 0 and 1 when *position*'s value is 40. Such function is the following:

$$f(x) = \frac{1+a}{1+c\cdot(x-x_1)^2} - a$$

where *a*=0.104 and *c*=0.006.

The graph of *f(x)* is depicted in Fig.4.

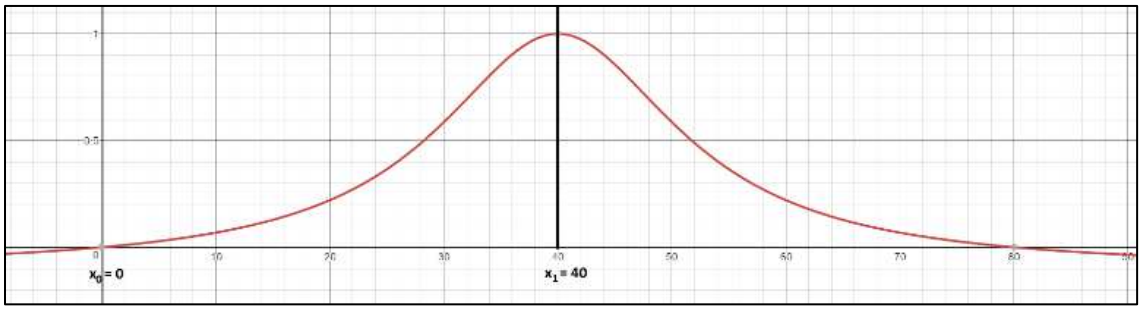

Fig.4 The graph of $f(x) = \frac{1+a}{1+c\cdot(x-40)^2} - a$ for the normalization of aperture knob's position (a=0.104, c=0.006)

Similar are the normalizations for the numeric values of other changeable features.

4.3. Success Rate

Having specified the various individual scores, we proceed with the definition of their combined score, namely the *success rate*.

The *success rate* is defined as the weighted average percentage of all individual scores:

$$success\ rate \leftarrow \frac{\sum_{k=1}^{n} w_k \cdot x_k}{\sum_{k=1}^{n} w_k} \qquad (1)$$

where *n* is the number of steps required for the completion of the procedure (16 in our microscoping procedure), and w_i the predefined weight of the *i*-indexed individual score. As the individual scores cannot exceed the value of 1, it is guaranteed that neither can the *success rate*

be greater than 1. Nevertheless, the user is provided with a re-scaled *success rate* in the 0-100 range, for user-friendliness purposes.

4.4. Penalty Points

In practice, the *success rate* measures the "distance" of the various features' values from their optimal values, i.e., the values taken when the procedure in question has been successfully completed. What it does not measure though is the *order* in which those optimal values were achieved; in other words, in which order the necessary steps were made. That is made possible with the use of the *penalty points* metric. In brief, *penalty points* increase in value whenever an action is made in the wrong order. The description of the respective algorithm, however, is beyond the scope of this paper.

4.5. Aggregate Score

Upon completion of a play session in Evaluation Mode, the user is prompted with *aggregate score* which, like *success rate*, ranges from 0 to 1, but is also affected by other factors which are multiplied with the latter. Those factors are penalty points, briefly-mentioned in the previous section, *Δtime*, indicating the number of seconds passed from the beginning of the session until its ending minus the minimum time required for the completion of the experimental procedure, and *resetting rate*, measuring the extent to which the various instrument components have been reset to their original state.

A basic formula of *aggregate score* combining all of the afore-mentioned factors and ranging from 0 to 1 is:

$$aggregate\ score \leftarrow e^{-\frac{penalty\ points}{\beta}} \cdot e^{-\frac{|\Delta time|}{\gamma}} \cdot success\ rate \cdot resseting\ rate$$

where *β* and *γ* are positive constants.

5. Machine Learning for the Reconfiguration of the Assessment Mechanism

In the previous section, we described the assessment mechanism used for the evaluation of the human user's performance. Yet the weights w_i for the *success rate* have up to now been defined intuitively by a human expert. The most efficient way to overcome the obstacle of the intuitive weights is to use machine learning for their redefinition.

The *success rate* metric is a rather static one, meaning it takes into account only the final states of various objects and not the order of the steps taken for their achievement; for the particular case of the microscoping of a test specimen procedure, it includes 16 different weights, w_1 to w_{16}, each one for the respective feature state changes that need to be made. Nevertheless, even the expert-supplied weights cannot be considered as totally reliable, since the experts feel much more at ease to give an overall score as opposed to breaking it down. Hence the need for a machine learning technique to be used on those weights. We chose our first technique to be a Genetic Algorithm.

Afterwards, we set an Artificial Neural Network to be an alternate success rate measure in place of the weighted average, with its first layer units consisting of the individual scores, and we train it with the use of the Back-Propagation algorithm. While through GA training the weights w_i of the weighted average are being calibrated, through the BP training on the ANN, the calibration concerns the weights of the latter.

5.1. Genetic Algorithm

We remind the reader that *success rate* is given by formula (1) in section 4.3.

The weights w_1, w_2, … , w_n in *success rate* represent the importance of their respective individual score x_1, x_2, … , x_n. We define *weight vector w* as:

$$\vec{w} = (w_1, w_2, \dots, w_n)^T \quad \textbf{(2)}$$

For our GA, each weight consists of a *gene* and each weight vector of a *chromosome*.

At first, we randomly produce a *population* of *p* weight vectors, $\overrightarrow{w^1}$, $\overrightarrow{w^2}$, … , $\overrightarrow{w^p}$, or, in short, $\overrightarrow{w^i}$, where $i = 1 \dots p$.

Fig.5 gives an illustrative description of the genes, chromosomes and populations of our GA.

Specializing (2) for each weight vector, we get:

$$\overrightarrow{w^i} = \left(w_1^i, w_2^i, \dots, w_n^i\right)^T \quad \textbf{(3)}$$

Now reformulating (1) in vector terms, we get:

$$success\ rate \leftarrow \frac{\overrightarrow{w^i} \cdot \vec{s}}{\left\|\overrightarrow{w^i}\right\|_1}$$

where $\left\|\overrightarrow{w^i}\right\|_1$ is the first-degree norm of $\overrightarrow{w^i}$ vector, equal to $\sum_{k=1}^{n} w_k^i$.

The first generation of weight vectors is produced by creating a set of weight vectors, $\overrightarrow{w^1}$, $\overrightarrow{w^2}$, … , $\overrightarrow{w^p}$, with random values from 0 to 100 as their components.

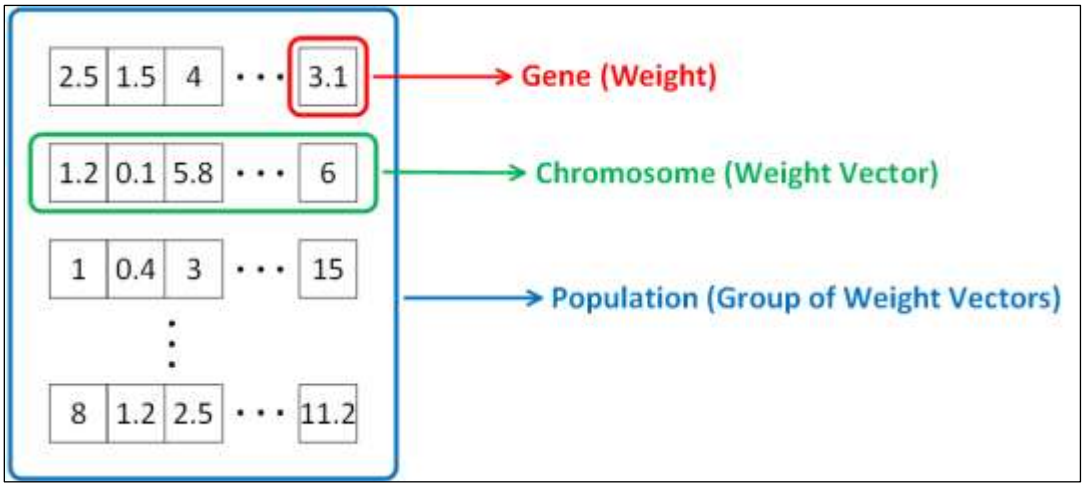

Fig.5 Genes, Chromosomes and Populations of our GA

The *fitness function* of our GA (probabilistically) chooses which weight vectors will survive in the next generation, as a whole or through crossover with other weight vectors. It must therefore take into account how accurately the *success rate* produced by a particular weight vector can approximate the score given by the expert on all of the examples of the training data set.

After a play session is completed in Training Mode, the computer calculates the various individual scores x_i and the resulting *success rate* while the expert provides his or her own evaluation.

For the individual scores, we create a *score vector* $\vec{x}$, similar to the *weight vector* of (2):

$$\vec{x} = (x_1, x_2, \dots, x_n)^T$$

Assuming for simplicity reasons that only one session has been played, so there is only one *score vector*, $\vec{x}$, and denoting expert's score for this session as $ES(\vec{x})$, we will proceed by defining a generic form of *fitness function*:

$$Fitness_{generic}(\overrightarrow{w^i}) = 1 - |ES(\vec{x}) - \frac{\overrightarrow{w^i} \cdot \vec{x}}{\|\overrightarrow{w^i}\|_1}| \qquad \textbf{(4)}$$

The *generic fitness function* gets its maximum value, 1, when the produced *success rate*, $\frac{\overrightarrow{w^i} \cdot \vec{x}}{\|\overrightarrow{w^i}\|_1}$, equals the expert's score, and its minimum value when those two quantities are as "far" as possible from each other.

Of course, a single *score vector*, i.e., a single play session or, in machine learning terms, a single training data point, does not suffice for our GA. We therefore play several sessions and are provided with a series of score vectors, $\overrightarrow{x^j}$, $j = 1 \dots h$, and their respective *expert scores*, $ES(\overrightarrow{x^j})$. Adjusting the generic *fitness function* in (4) for each *score vector* $\overrightarrow{x^j}$, we have:

$$Fitness_j(\overrightarrow{w^i}) = 1 - |ES(\overrightarrow{x^j}) - \frac{\overrightarrow{w^i} \cdot \vec{x}}{\|\overrightarrow{w^i}\|_1}| \qquad \textbf{(5)}$$

An obvious *overall fitness function* of weight vector $\overrightarrow{w^i}$ is the average of the various *fitness functions* described in (5). Thus, we define:

$$Fitness(\overrightarrow{w^i}) = \frac{\sum_{j=1}^{h} Fitness_j(\overrightarrow{w^i})}{h}$$

$$Fitness(\overrightarrow{w^i}) = \frac{\sum_{j=1}^{h} \left[1 - \left| ES(\overrightarrow{x^j}) - \frac{\overrightarrow{w^i} \cdot \overrightarrow{x^j}}{\|\overrightarrow{w^i}\|_1} \right| \right]}{h}$$

$$Fitness(\overrightarrow{w^i}) = 1 - \frac{\sum_{j=1}^{h} \left| ES(\overrightarrow{x^j}) - \frac{\overrightarrow{w^i} \cdot \overrightarrow{x^j}}{\|\overrightarrow{w^i}\|_1} \right|}{h}$$

where *h* is the number of the different score vectors, or training examples.

The generic fitness function (hence the overall fitness function, too) defined above is negative linear. We also define three alternative generic fitness functions:

1. negative quadratic: $1 - (ES(\vec{x}) - \frac{\overrightarrow{w^i} \cdot \vec{x}}{\left\| \overrightarrow{w^i} \right\|_1})^2$
2. negative exponential: $e^{-\lambda \cdot \left| ES(\vec{x}) - \frac{\overrightarrow{w^i} \cdot \vec{x}}{\left\| \overrightarrow{w^i} \right\|_1} \right|}$
3. inverse: $\frac{1}{\lambda \cdot \left| ES(\vec{x}) - \frac{\overrightarrow{w^i} \cdot \vec{x}}{\left\| \overrightarrow{w^i} \right\|_1} \right|}$

where λ is a constant greater than 1.

The overall fitness function for the three new generic fitness functions is produced similarly to the first one's. The user may use any of those generic fitness functions.

The new generation to be bred must contain the same number of weight vectors, i.e., *p*. Whereas a part of them will be "duplicates" of weight vectors from the current generation (*selection* operation), the other part will be "reproduced" by random pairs among those selected weights vectors (*crossover* operation).

At first, we calculate the *selection probability* for each weight vector of the current generation:

$$\Pr(\overrightarrow{w^i}) = \frac{Fitness(\overrightarrow{w^i})}{\sum_{k=1}^{p} Fitness\left(\overrightarrow{w^k}\right)}$$

We now define a ratio *r* representing the proportion of replaceable weights (*r* being static for all generations), i.e., the number of weight vectors which will not survive into the new generation. Then we apply the *selection* operation, i.e., we select $(1 - r) \cdot p$ weight vectors from the current generation according to their selection probability and "copy" them into the new generation.

At the *crossover* operation, we choose $\frac{r \cdot p}{2}$ pairs of weight vectors from the current generation (including those who were selected before), with respect to their *selection probability*, to crossover and put their offspring (two for each pair) into the new generation.

Last comes the *mutation* operation. Setting mutation rate to be *m* (being static for all generations, like *r*), we choose, with uniform probability, *m* percent of the weight vectors that have been created in this new generation to be mutated. We have not defined just one type of mutation but three; doubling of a gene (weight value); halving of a gene; and permutation of two genes within a chromosome (weight vector). The user may choose any one of them, like he or she chooses a generic fitness function.

This second generation of weight vectors that has been produced is then used with the exact same score vectors (i.e., the same playing outcomes) and the same expert's success rate (i.e., the same evaluation by the human expert for each respective playing outcome) that the first generation was used with; afterwards, it undergoes the same operations of selection, crossover and mutation producing the third generation, and so on.

We finally need to specify a termination condition for our GA. We could set the latter to halt when the maximum of $\{Fitness(\overrightarrow{w^i}), 1 \leq i \leq p\}$ at a generation becomes greater than a threshold (e.g., 0.95). However, if a particular limit of generations were reached without any of the weight vectors of the last generation satisfying the termination condition, the algorithm would need to be restarted or the threshold to be reduced. Thus, in order to secure that our GA eventually ends, we set the termination condition to be an explicit number of generations, specified each time by the user (Zafeiropoulos & Kalles, 2019, 2022).

5.2. Artificial Neural Network

The ANN we have designed is a three-layer one; it contains *k+1* input units in the first layer (*k* units for each individual score x_k and a bias), 3 units in the second (hidden) layer and one output unit in the third layer, while its squashing function for hidden layer and output units is the sigmoid one ($\frac{1}{1+e^{-x}}$). Each unit of the hidden layer is connected to each unit of the first layer through a different weight. In the same fashion, different weights connect the sole unit of the output layer to the units of the hidden layer. Fig.6 displays the structure of our ANN.

The success rate is set to be equal to the *rescaled* value of the output layer unit. The rescaling of the ANN output value into [0,1] interval is necessary because, unlike the weighted average, the ANN might not produce a value within the said field. For the rescaling two outputs need to be calculated; one output produced by the ANN with all individual scores being 0, or the score vector being <0,0,…,0>, and another one produced with all input values being 1, or the score vector being <1,1,…,1>. The value produced with the <0,0,…,0> vector, i.e., when the user has not yet performed any action and is at their initial state, even though it may not be 0 itself, is set to correspond to a success rate equal to 0. We name this value $output_{zero}$. Likewise, the value produced with the <1,1,…,1> vector, i.e., when the user has successfully performed every necessary action and reached the final state, is set to correspond to a success rate equal to 1. We name this value $output_{one}$. The rescaling is done through a linear function $y = a \cdot x + b$, where *a* and *b* coefficients are calculated to be $\frac{1}{output_{one} - output_{zero}}$ and $-a \cdot output_{zero}$ respectively. We must note here that the output is not rescaled into the [0,1] interval *before* or *during* training but only *after* the latter is completed and now the ANN is used for the user's evaluation.

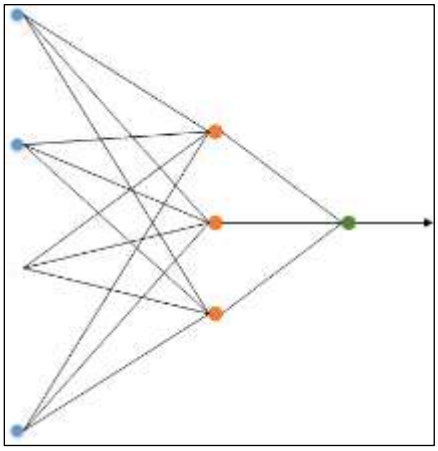

Fig.6 The 3-layer ANN with 3 units in hidden layer, used for the assessment of user's performance

The ANN training is done with the prototypical BP algorithm (Mitchell, 1997). Its weights are initialized with random values from -0.05 to +0.05 (Zafeiropoulos & Kalles, 2022).

6. Training Results

As shown in section 5, rater training is done in two ways; using a Genetic Algorithm in case the *success rate* is computed as a weighted average, and using Back-Propagation in case the *success rate* is computed by an Artificial Neural Network. The training results are then evaluated with the use of the Mean Squared Error measure (Zafeiropoulos & Kalles, 2022). Before getting into any details though, we shall explore some aspects of our data collection stage.

6.1. Training Data

Every training data point consists of the individual scores achieved by a human user in a play session plus the estimation of their achieved success rate as given by a human expert, along with the identifier of the expert and the 'Low', 'Medium' or 'High' classification of the session in terms of the user's performance.

For the purpose of our research, we asked 4 biology experts at the Hellenic Open University to evaluate 60 play sessions, which were conducted by one of the authors of the paper. In particular, each expert evaluated 15 play sessions. Among those 15 sessions, 5 were played as 'Low' by the paper's author based on his knowledge of the domain, as gained during development, and classified as such by the particular expert; 5 were played and classified as 'Medium'; and the other 5 were played and classified as 'High'. We therefore have in total 20 sessions classified as 'Low', 20 as 'Medium' and 20 as 'High'.

We should note here that the range of our data collection does not aim to definitively calibrate the success rate weights to actually usable values; instead, its purpose is to assist the investigation of which machine learning techniques seem to be suitable for this setting of the scoring mechanism. Essentially, we set the ground so that a statistically sound designed experiment can be then deployed to produce actually usable weights.

6.1.1. Training and Testing on Various Data Set Groups

In order to conduct the training and testing, we form various groups of our training examples. As we briefly mentioned in previous chapters, a training data set consists of a single play session – specifically the individual scores achieved upon the completion of the session – as well as the particular expert's evaluation (expert's score) for that.

Initially, we run an "optimistic" case, by applying resubstitution on all examples, i.e., training our system on all data points and testing the training results also on all data points, and getting the respective Mean Squared Error (MSE). Then, we drop the "optimistic" assumption and we apply cross-validation again on all data points, i.e., training on some of the data points and testing the trained models on the rest of the data points.

Apart from resubstitution on all examples, we also run separate resubstitutions on the examples generated by each expert. Finally, we run several cross-validation sessions, varying the type of data partition; an initial 3-fold cross-validation with each fold consisting of only "Low" or

"Medium" or "High" examples; and a 4-fold cross-validation, with each fold consisting of data points from the same expert.

6.2. Results in Fine-Tuning the Success Rate Weights with a Genetic Algorithm

We say that a GA training combination *converges* if the MSE in the respective graph decreases as the number of generations increases.

We tried numerous combinations of generic fitness functions and permutation methods with various different values for the number of population members, the crossover rate, the mutation rate and the number of generations. For most of the cases, no convergence was achieved. A typical behavior of no convergence is the one of negative linear generic fitness function with permutation of two genes as a mutation method, shown in Fig.7.

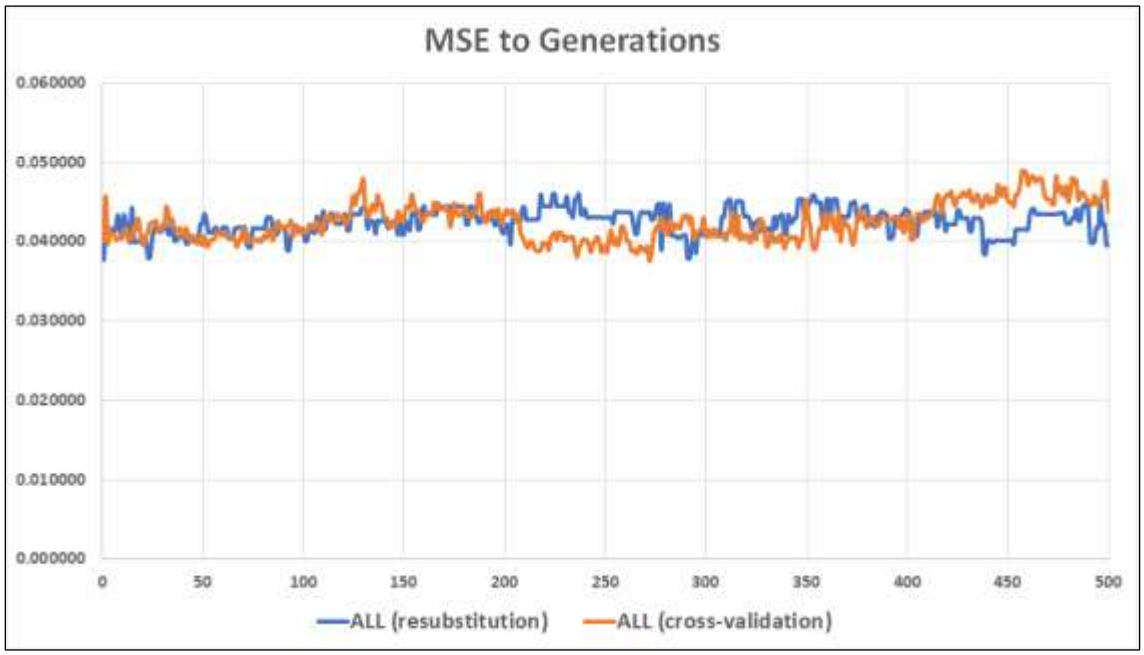

Fig.7 MSE to Generations Graphs for GA – Resubstitution and Cross-Validation on all data sets (Population Members: 100, Crossover Rate: 0.9, Mutation Rate: 0.01, Generations: 500, Generic Fitness Function: negative linear, Mutation Method: permutation of two genes)

Convergence was achieved for a very few cases, one of them being the inverse generic fitness function with halving of a gene as a mutation method. There, we have very small convergence of around 0.013 and 0.004 magnitude for resubstitution and cross-validation, respectively, as shown in Fig.8.

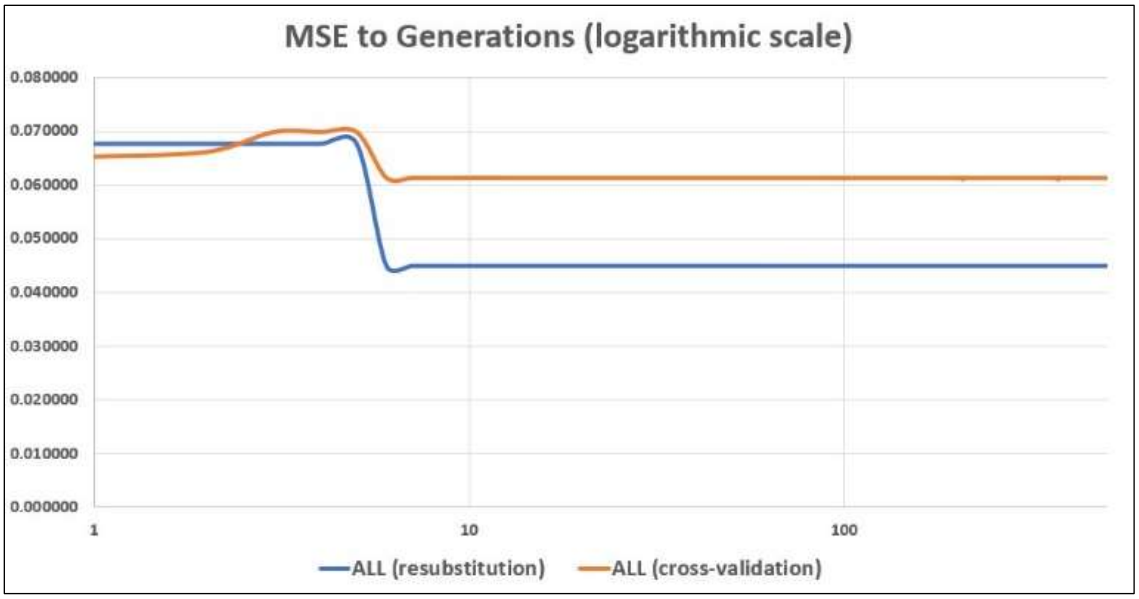

Fig.8. MSE to Generations Graphs (logarithmic scale) for GA – Resubstitution and Cross-Validation on all data sets (Population Members: 100, Crossover Rate: 0.9, Mutation Rate: 0.03, Generations: 500, Generic Fitness Function: inverse, Mutation Method: halving of a gene)

In addition to the convergence being rare, the weight vectors resulting from training seem unrealistic. For example, in the latter case, the produced weight vector is:

$$< 4.69, 2.03, 27.01, 8.85, 23.59, 8.97,$$

$$22.01, 7.26, 1, 25,$$

$$17.03, 20.53,$$

$$3.02, 3.09, 24.64, 21.39 >$$

while the intuitive weight vector is:

$$< 10{,}10, 5, 10, 2, 10,$$

$$5, 5, 2, 2,$$

$$10, 1,$$

$$8, 8, 8, 8 >$$

with the similarity between the two vectors being 63.83%.

One notices several inconsistencies in the new weight vector such as the weight for testing the specimen holder knob being 25 (2^{nd} row, 3^{rd} position) while the weight for testing the stage knob being 1 (2^{nd} row, 4^{th} position). That gives us ample indications that our weighted average model of evaluation is incomplete and needs to be enhanced. In the next chapter, we deal with a particular enhancement of our evaluation algorithm and examine its response to GA training.

6.3. Results in Estimating the Success Rate Score Function with an Artificial Neural Network

In a similar fashion to GA training, we say that an ANN training combination *converges* if the MSE in the respective graph decreases as the number of epochs increases.

In the ANN training case, in contrast to the GA, convergence is achieved in approximately 80% of cases of our data sets groupings. Below we show some indicative examples out of the performed experiments.

Fig.9 shows the MSE to Epochs graphs (epochs: 2000) for resubstitution and cross-validation on all data sets; in the resubstitution case the graph converges quite early, while in the cross-validation case the convergence becomes maximum in approximately the 400th epoch and, thereafter, starts to gradually, yet slowly, deteriorate.

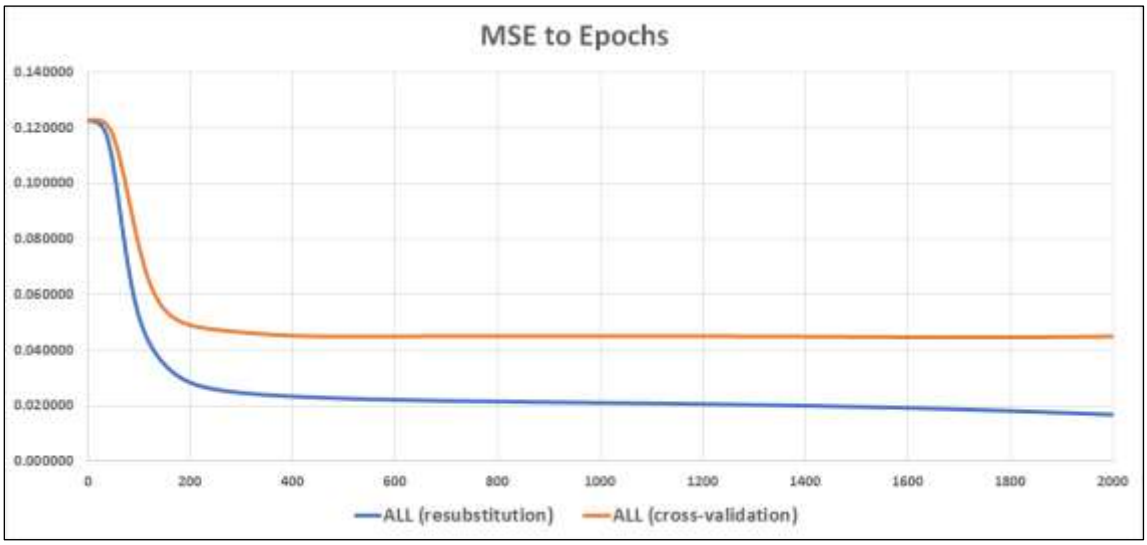

Fig.9 MSE to Epochs Graph for training of 3-layer ANN with 3 units in hidden layer – Resubstitution and Cross-Validation on all data sets (Epochs: 2000)

The same behavior is exhibited in the resubstitution and cross-validation cases within a single expert's data sets. Fig.10 shows the respective MSE to Epochs graphs for one of our experts for the same epochs.

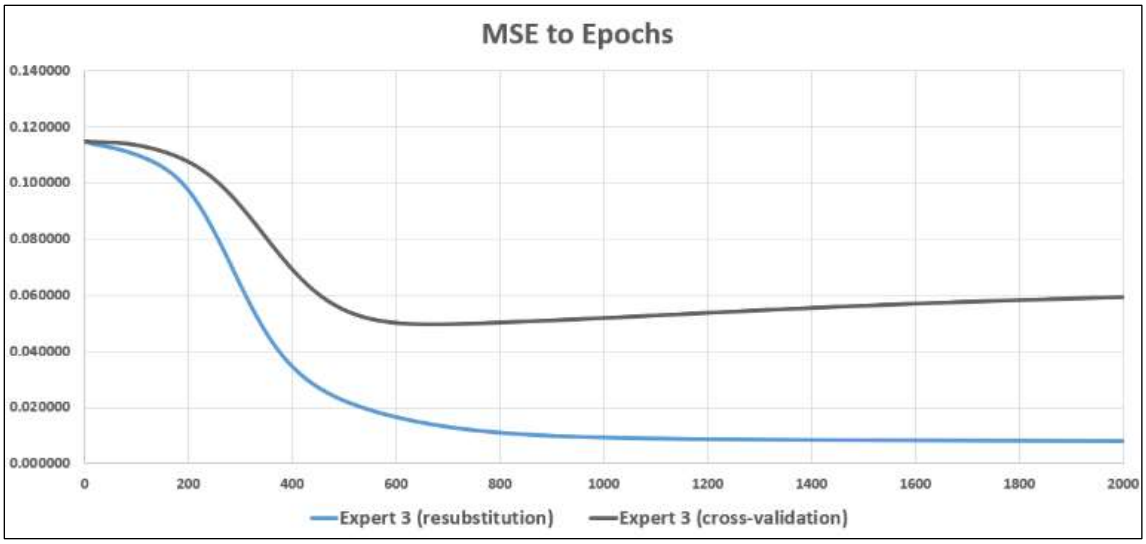

Fig.10 MSE to Epochs Graph for training of 3-layer ANN with 3 units in hidden layer – Resubstitution and Cross-Validation within one of our expert's data sets (Epochs: 2000)

Similarly, cross-validation among experts also converge within the first 400-800 epochs, then slightly diverge and after some point start converging again, as shown in Fig.11.

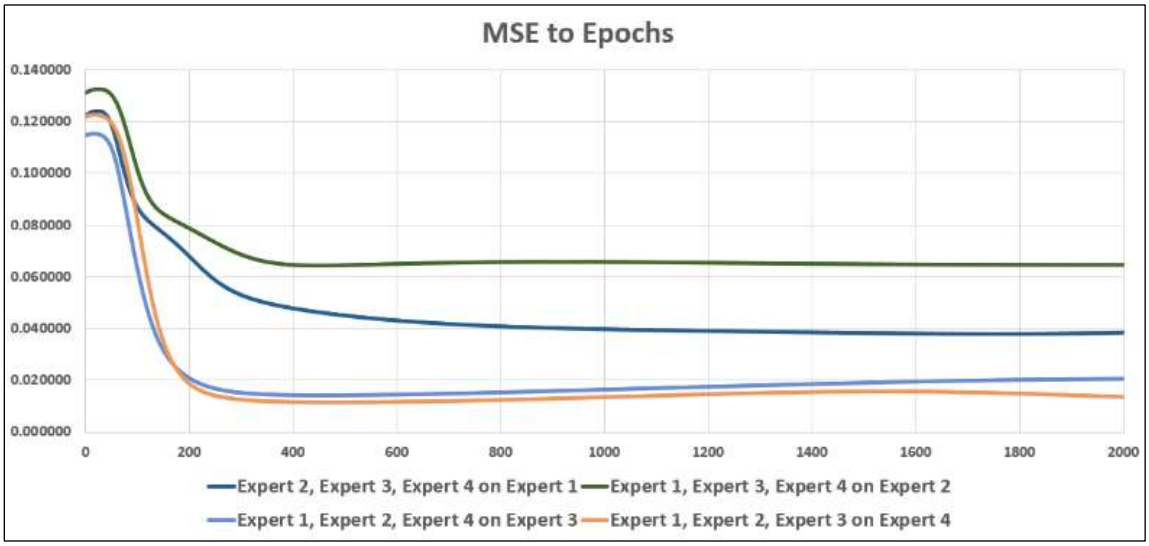

Fig.11 MSE to Epochs Graph for training of 3-layer ANN with 3 units in hidden layer – Cross-Validation among experts (Epochs: 2000)

On the contrary, convergence to a higher than the original MSE or no convergence at all is achieved in the cross-validation among different classifications, as shown in Fig.12.

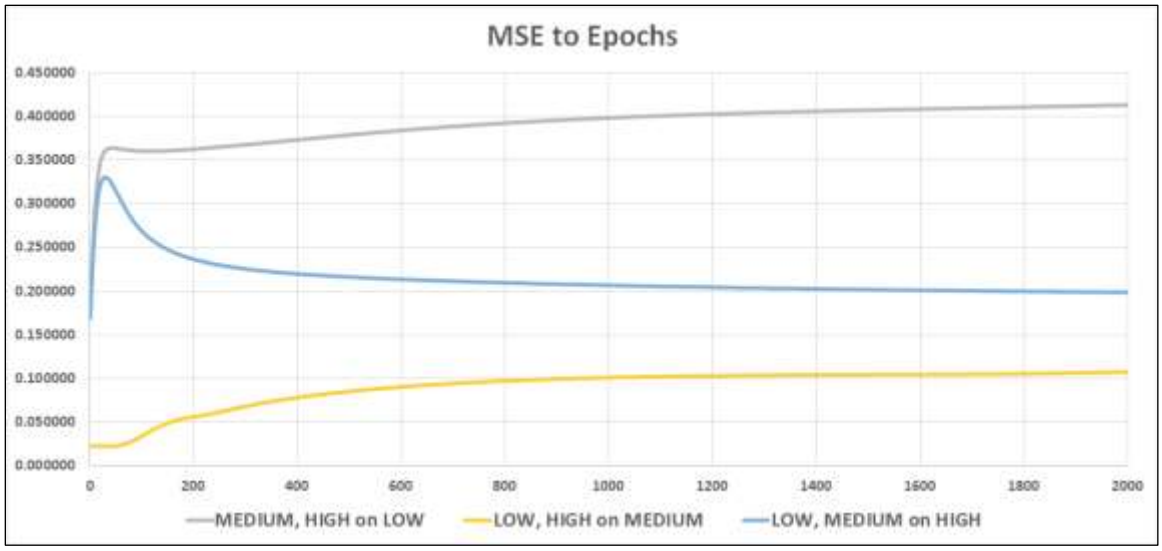

Fig.12 MSE to Epochs Graph for training of 3-layer ANN with 3 units in hidden layer – Cross-Validation among classifications (Epochs: 2000)

6.3.1. Inconsistencies between Step-by-Step and Global Estimation of the Success Rate Score Function

Despite the fact that BP training converges in the vast majority of cases, the produced ANN does not correspond to a realistic *success rate*, i.e., a realistic output value after being rescaled to the [0,1] interval. In fact, some actions may result into an unrealistic increase, or even decrease, of *success rate*. In our microscoping test-case and for 2000 epochs, we observe that the success rate decreases from step 3 to step 6, as shown in Fig.13, while normally, it should increase for every new step made, hence from step 3 to step 6 as well. In Fig.13, we also see a slight decrease from step 12 to step 13, instead of the respective expected increase.

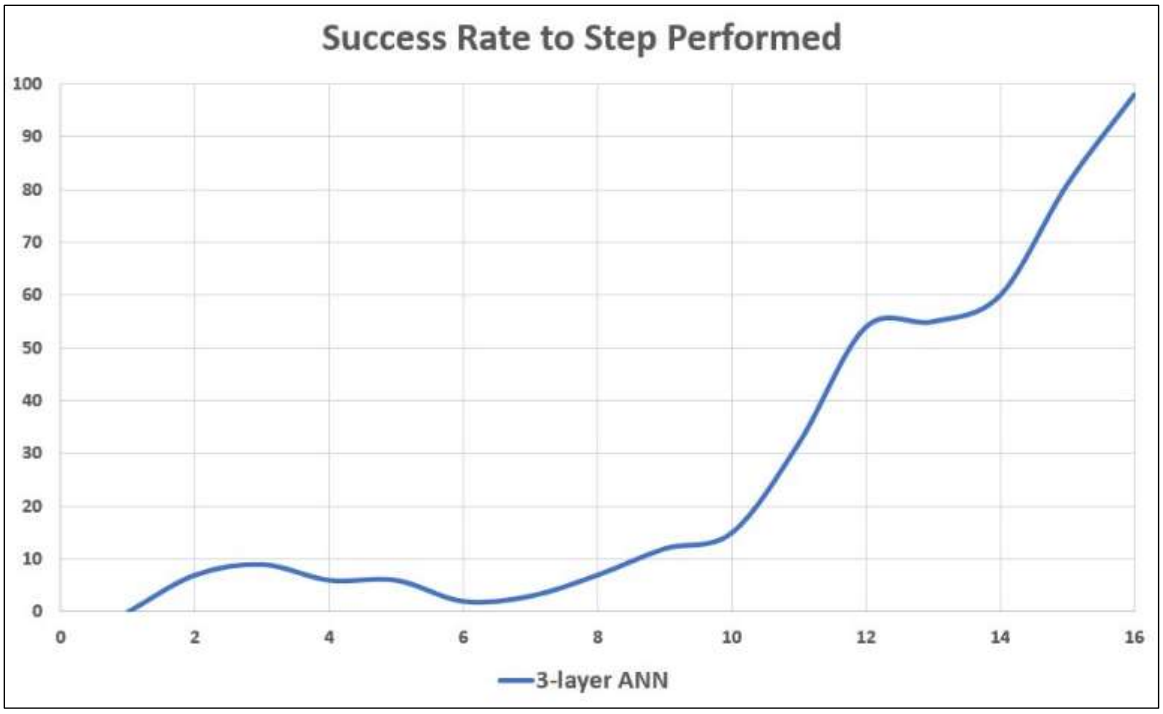

Fig.13. Success Rate Increase to Performed Step Graph for training of 3-layer ANN with 3 units in hidden layer (Epochs: 2000)

This kind of inconsistencies are common in the training results for all our cases of data sets groupings. It is therefore reasonable to conclude that our ANN is incomplete in terms of the performance evaluation it makes. In the next section, we will discuss a particular improvement for our ANN, similar to the one which will be suggested for the GA.

7. Hierarchical Learning with a Domain Context

As we saw in the previous chapter, the Genetic Algorithm used for the calibration of the weights of the weighted average type of *success rate* generally did not converge, i.e., there was no virtual decrease in the MSE measure after training. In the exceptional cases that we had small convergence, the weights produced by the GA were unrealistic and did not contribute to a reasonable scoring. That, inevitably, makes us question whether our weighted average measure of *success rate* is appropriate at all.

In this section, first we further explore this issue by modifying *success rate* into a hierarchical fashion and training the GA with it. Next, in a similar fashion, we attempt the hierarchical reconstruction of the Artificial Neural Network used as our alternate *success rate* measure.

7.1. Defining the Hierarchy

The microscoping procedure, as described in previous sections, consists of 16 actions which need to be carried out and each of which has been assigned a weight, according to its significance, to reflect its contribution to the total score. However, even though those 16 weights can be different from each other, that quantitative difference is the only one we are taking into account; i.e., the weighted average into which they contribute is *flat*, in the sense that actions are considered to

be placed at the same conceptual level, albeit with a difference in significance. This, obviously, is an over-simplification, at least for most of them. For example, testing a particular knob is just a sub-step of the sub-procedure of testing all knobs while microscoping with a particular objective lens is a sub-step of the sub-procedure of microscoping with all lenses; thus, it seems rather unreasonable to create an artificial relationship between these two steps (testing a knob and using a particular objective lens) instead of allowing that relationship to stand between the actual sub-procedures they belong to.

We have therefore valid reasons to suspect that the non-convergence and the unrealistic weights produced by training were a result of the actions being taken as being conceptually equivalent. Thus, we shall attempt to redefine the success rate by dividing the microscoping procedure into the 4 different sub-procedures shown in Table 2 and allowing that way for more conceptually-relevant grouping to contribute to scoring.

Table 2. 16-step microscoping procedure divided into 4 sub-procedures

1. Set microscope 1.1. Connect the microscope into the socket 1.2. Turn the microscope light on 1.3. Set light intensity to 2 3rds of maximum 1.4. Set iris fully open 1.5. Lift the condenser lens to the top position 1.6. Set lens 4X active
2. Test microscope knobs 2.1. Test coarse focus knob 2.2. Test fine focus knob 2.3. Test stage knob 2.4. Test specimen holder knob
3. Prepare actual microscoping 3.1. Put test specimen on stage 3.2. Enter microscope mode
4. Perform actual microscoping 4.1. Focus with lens 4X 4.2. Focus with lens 10X 4.3. Focus with lens 40X 4.4. Focus with lens 100X

As one sees, the steps of each sub-procedure are conceptually more relevant to each other than to the steps of the other sub-procedures; hence, the individual scores representing the respective feature value changes, too.

7.2. Hierarchical Weighted Average

With this structure in mind, we define a *hierarchical* weighted average for the success rate, i.e., a weighted average of the weighted averages of the individual scores of each sub-procedure. The formula would be:

$$hierarchical\ success\ rate \leftarrow \frac{\sum_{t=1}^{n_s}\left(v_t \cdot \frac{\sum_{k=1}^{n_t} w_{t,k} \cdot x_{t,k}}{\sum_{k=1}^{n_t} w_{t,k}}\right)}{\sum_{t=1}^{n_s} v_t}$$

where:

- $w_{t,k}$, $x_{t,k}$ are the weights and individual scores for the k-th action of the t-th sub-procedure, respectively
- n_t is total number of actions contributing to the t-th procedure
- v_t is the weight for the t-th sub-procedure
- n_s is the total number of sub-procedures

As a matter of convention, we refer to the weights v_t as *external weights*.

7.2.1. Training with Genetic Algorithms

We run the same Genetic Algorithm again, this time with the success rate being the hierarchical weighted average; the only change done is in equation (4) of section 5.1, with the substitution of the flat weighted average ($\frac{\overrightarrow{w^i}.\vec{x}}{\left\|\overrightarrow{w^i}\right\|_1}$) with the hierarchical one defined here.

The results of "hierarchical training" seem better than those of their "flat" counterpart (e.g., Fig.7 and Fig.8of section 6.2) as we achieve convergence in more trainings and for more combinations of parameters. We demonstrate some of the results in Fig.14, Fig.15 and Fig.16.

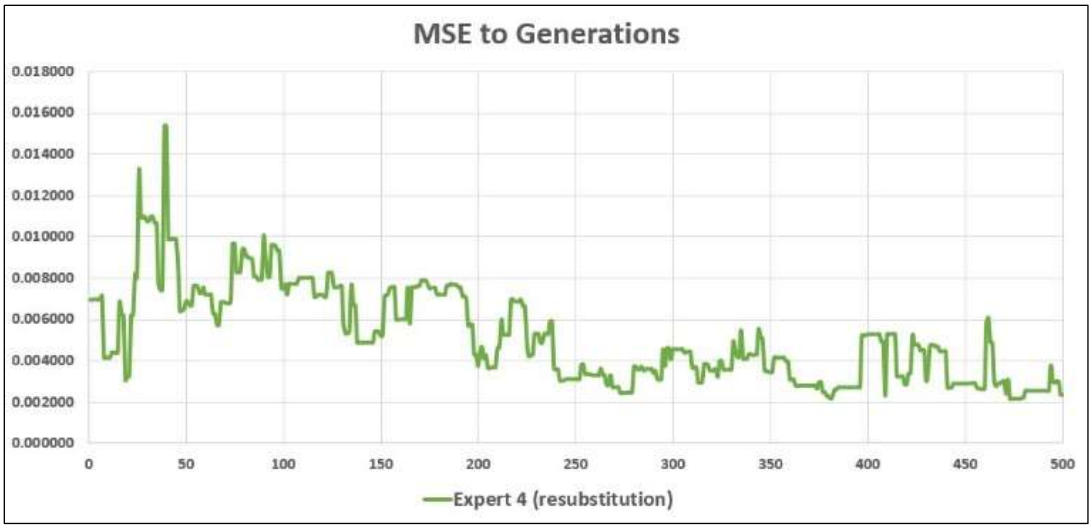

Fig.14 MSE to Generations Graphs for GA – Resubstitution on Expert 4 (Population Members: 100, Crossover Rate: 0.2, Mutation Rate: 0.3, Generations: 500, Generic Fitness Function: inverse, Mutation Method: doubling of a gene)

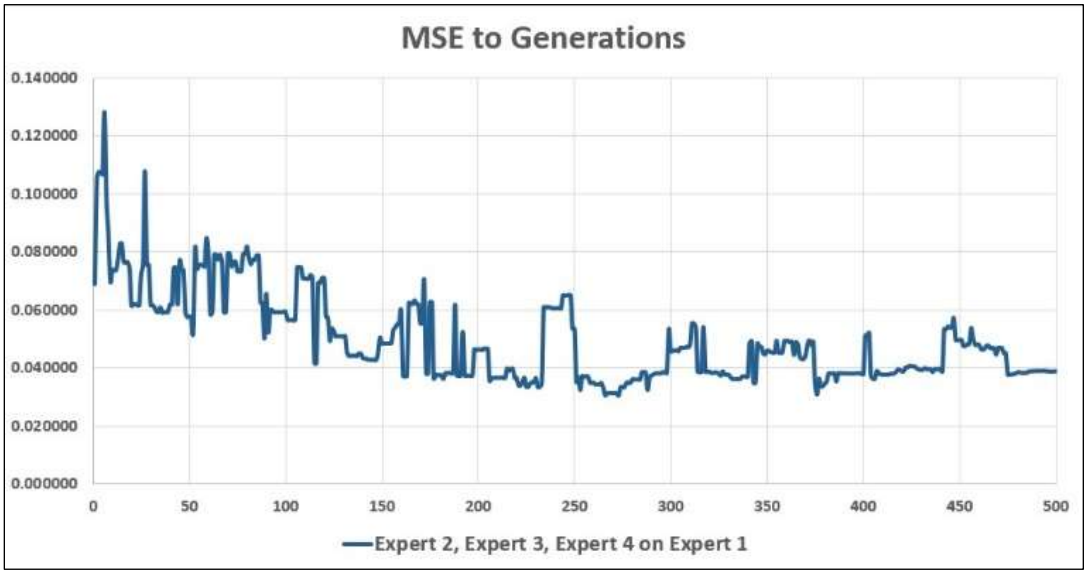

Fig.15 MSE to Generations Graphs for GA – Training on Experts 2, 3 and 4 and testing on Expert 1 (Population Members: 100, Crossover Rate: 0.5, Mutation Rate: 0.5, Generations: 500, Generic Fitness Function: negative exponential, Mutation Method: halving of a gene)

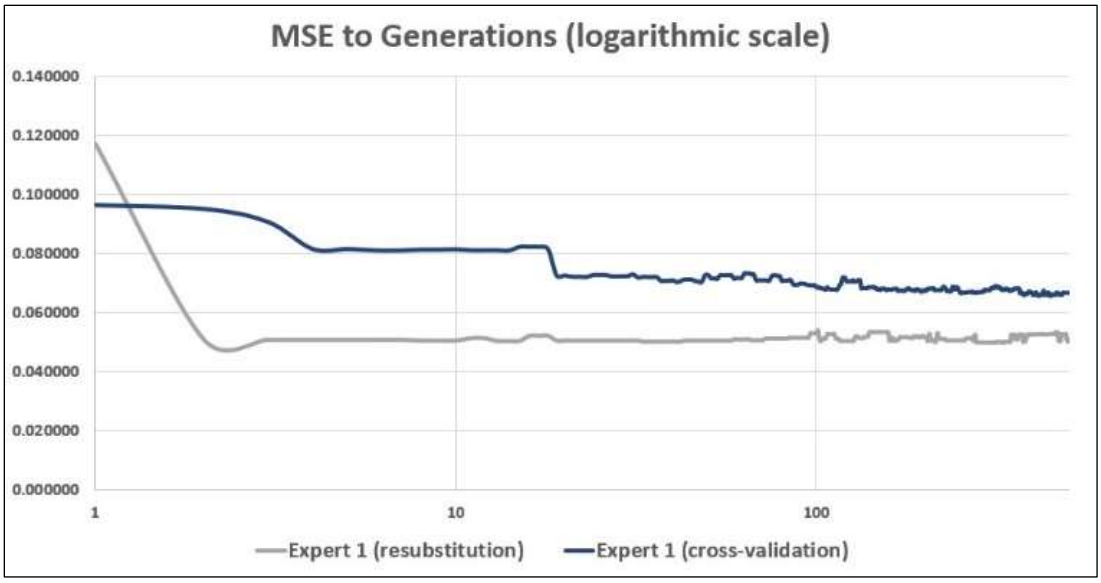

Fig.16 MSE to Generations Graphs for GA – Resubstitution and Cross-Validation on Expert 1's data sets (Population Members: 100, Crossover Rate: 0.6, Mutation Rate: 0.2, Generations: 500, Generic Fitness Function: inverse, Mutation Method: doubling of a gene)

However, as in the flat weighted average case, the resulting weights are not realistic. For example, in the last paradigm shown in Fig.16, training on Expert 1's data sets results into the following weight vector (the weights corresponding to each sub-procedure are included in the first four pair of brackets while the fifth one consists of the external weights):

$$< \{1, 1.11, 1.35, 2.85, 4.16, 1.92\},$$

$$\{3.45, 36.62, 1, 46.41\},$$

$$\{1.1, 1\},$$

$$\{115.96, 1, 11.34, 8.93\},$$

$$\{3.23, 1, 2, 2.22\} >$$

The similarity of the latter weight vector to the intuitive one is calculated to be only 35.09%. One also sees a big discrepancy between the weights regarding the sub-procedure of knob testing; testing the specimen holder knob has a weight of 46.41 while the weight for testing the stage knob is 1. Likewise, in the sub-procedure of focusing with the objective lenses, the weight for focusing with the 4X lens is 115.96 while the weights for focusing with any of the other objective lenses range from 1 to 11.34. Last but not least, the external weights have similar values which mistakenly treats each sub-procedure as more or less equally important.

As a consequence, we assume that despite our hierarchical weighted average securing convergence more often than our flat one, it is still not a sufficient model of evaluation. In other words, an overdue deep approach to modelling (hierarchical instead of flat) might be more productive in terms of accuracy but, eventually, less meaningful; i.e., trying to fine-tune the

performance might deteriorate the level of explainability. This concurs with how ANNs work, in general, where good performance does not necessarily enhance understanding.

7.3. Artificial Neural Network Restructuring

We will now try to incorporate our hierarchical redesign of microscoping procedure into the ANN used as our alternative *success rate* metric.

Since the microscoping procedure is split into 4 sub-procedures, a 3-layered ANN with 4 units in the hidden layer instead of 3 that we had until now seems applicable. The respective ANN of the one used so far but with 4 units in the hidden layer, conventionally called 'ANN-F4' (letter 'F' standing for "flat", i.e., non-hierarchical) is shown in Fig.17.

By the same convention, we name the previously-used ANN with 3 units in hidden layer (shown in Fig.6 of section 5.2) as 'ANN-F3'.

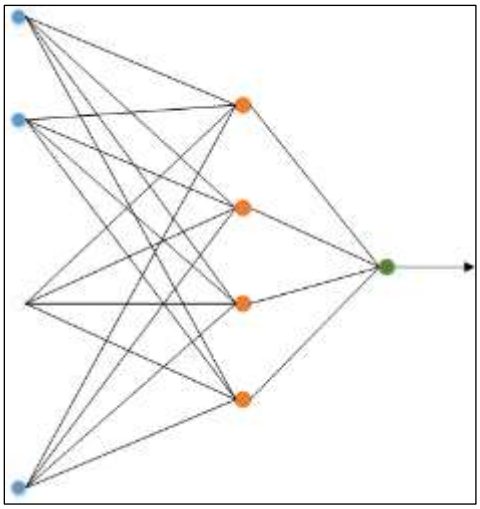

Fig.17 A 3-layer ANN with 4 units in hidden layer (ANN-F4)

As we said, ANN-F4, like ANN-F3, is non-hierarchical; its only difference from ANN-F3 is the number of units in the hidden layer. We will attempt to create a "hierarchical" ANN by modifying the interconnections of the various units between input and hidden layer of ANN-4F, taking into consideration the hierarchy of microscoping actions in section 7.1.

In that section, we divided microscoping procedure into 4 sub-procedures. Assuming that each unit of the hidden layer corresponds to each one of the sub-procedures, a hierarchical reconstruction of ANN-F4 would be to interconnect each unit of the hidden layer *only* with the respective individual scores from the input layers. The new hierarchical ANN, which we conventionally name 'ANN-H4' (letter 'H' standing for "hierarchical"), is shown in Fig.18.

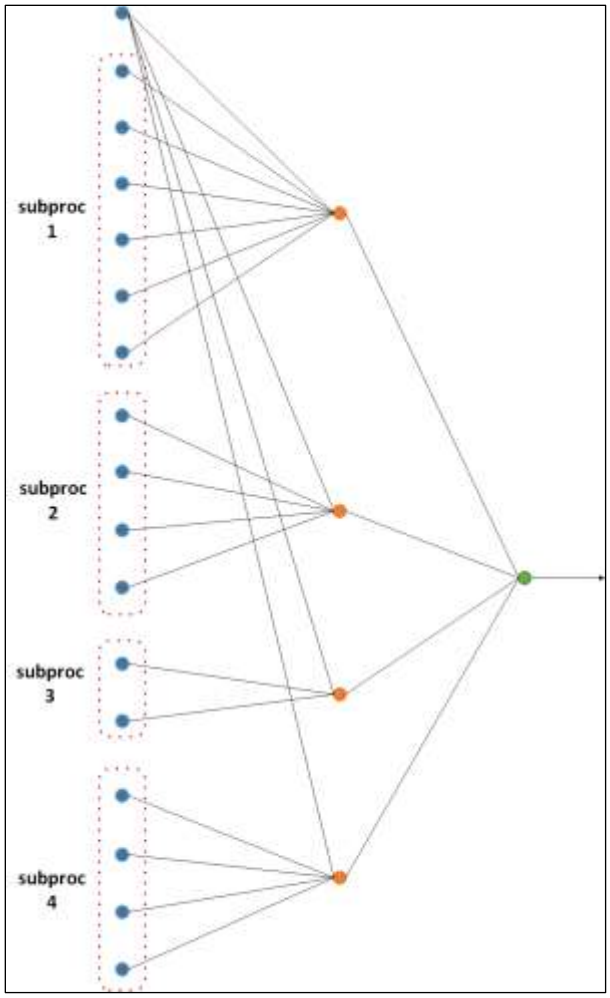

Fig.18 A hierarchical 3-layer ANN with 4 units in hidden layer (ANN-H4)

1.1.1 Artificial Neural Network Training

We will now train the two new ANNs we created (ANN-4F and ANN-4H) on the existing data sets and compare their results with each other but with the ones of ANN-3F as well.

Firstly, the training of ANN-4F gives us no better results that those of ANN-3F. While resubstitution for all training data sets gives almost identically convergent graphs for both ANNs and the same number of epochs (2000), cross-validation graph of ANN-4F behaves much worse than that of ANN-3F. Fig.19 shows ANN-4F graph starting to diverge at approximately the 300^{th} epoch, assumingly due to overfitting, while ANN-3F shown in Fig.6 of section 5.2, does not diverge at all.

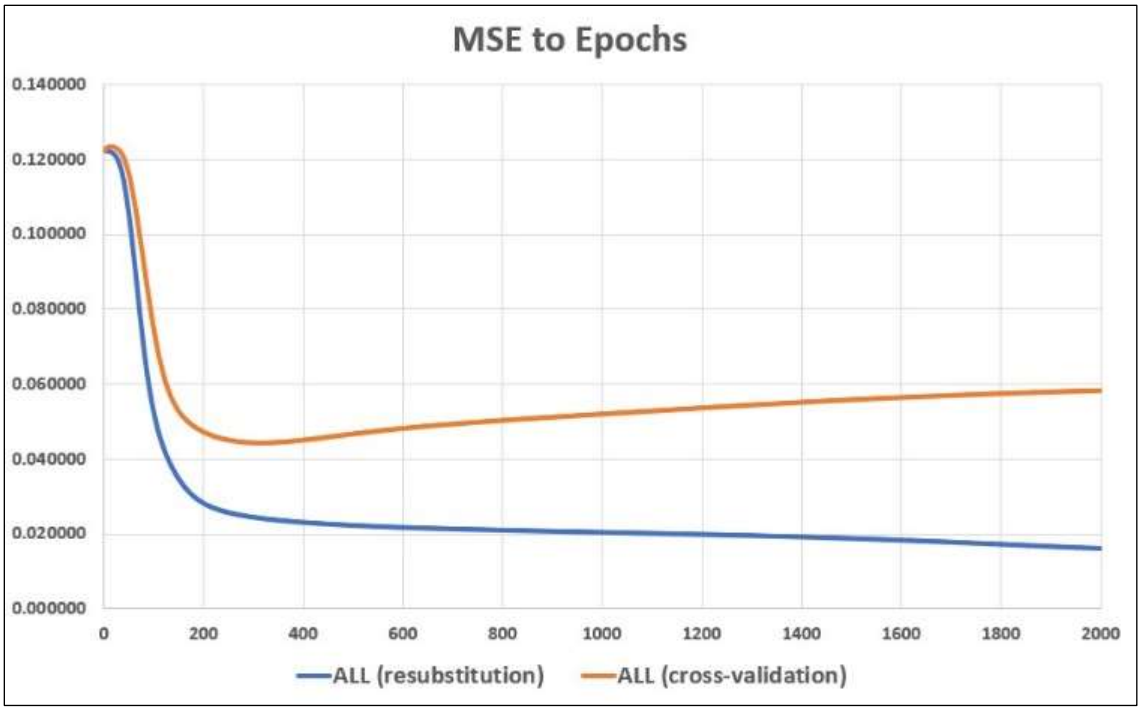

Fig.19 MSE to Epochs Graph for 3-layer "flat" ANN training with 4 units in hidden layer (ANN-4F) – Resubstitution and Cross-Validation on all data sets (Epochs: 2000)

Convergence is not better in the case of ANN-4H either. As Fig.20 shows, MSE is stabilized for both resubstitution and cross-validation for all training data sets at approximately 0.05; which consists of a worse outcome than in the resubstitution for all training data sets on ANN-4F as well as in both resubstitution and cross-validation for all training data sets on ANN-3F.

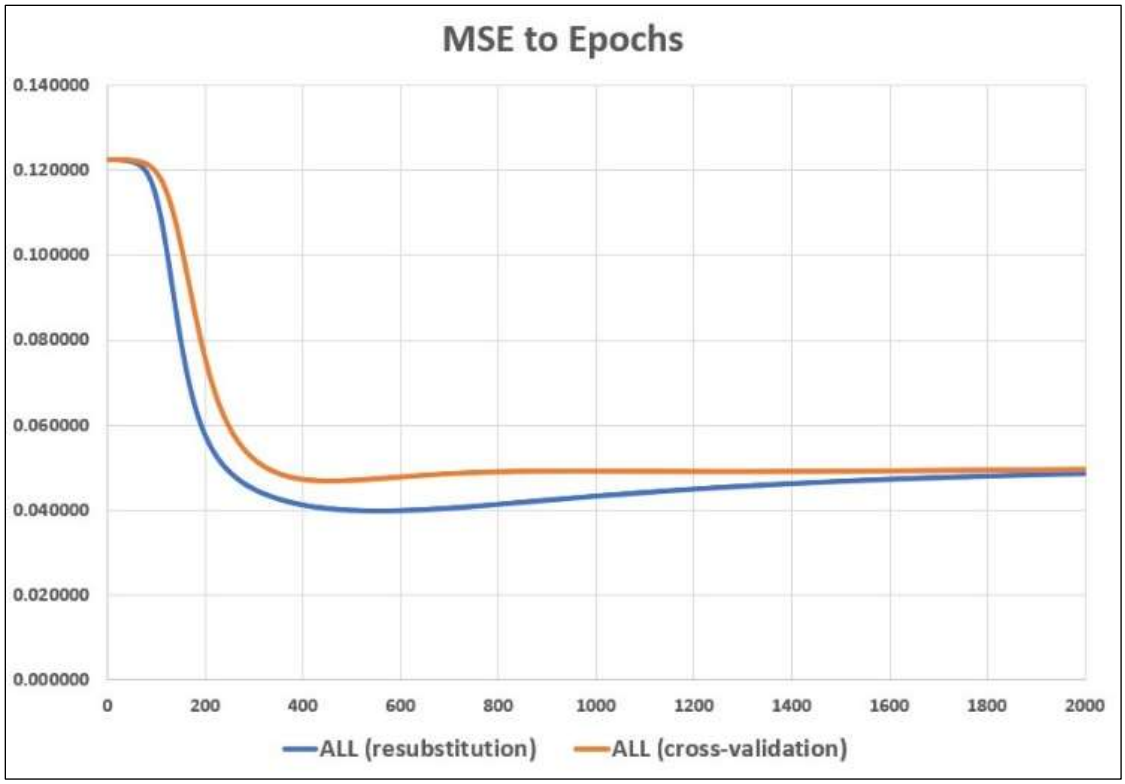

Fig.20 MSE to Epochs Graph for 3-layer "hierarchical" ANN training with 4 units in hidden layer (ANN-4H) – Resubstitution and Cross-Validation on all data sets (Epochs: 2000)

Despite not actually getting better results in terms of convergence with any of those new ANNs, we need to examine how the success rate changes according to the various steps performed. A comparison of the performance of all three ANNs is illustrated in Fig.21. The latter explicitly shows that ANN-4H, albeit producing worse results in terms of convergence, has generally a smoother *success rate* increase as well as the fewest decreases.

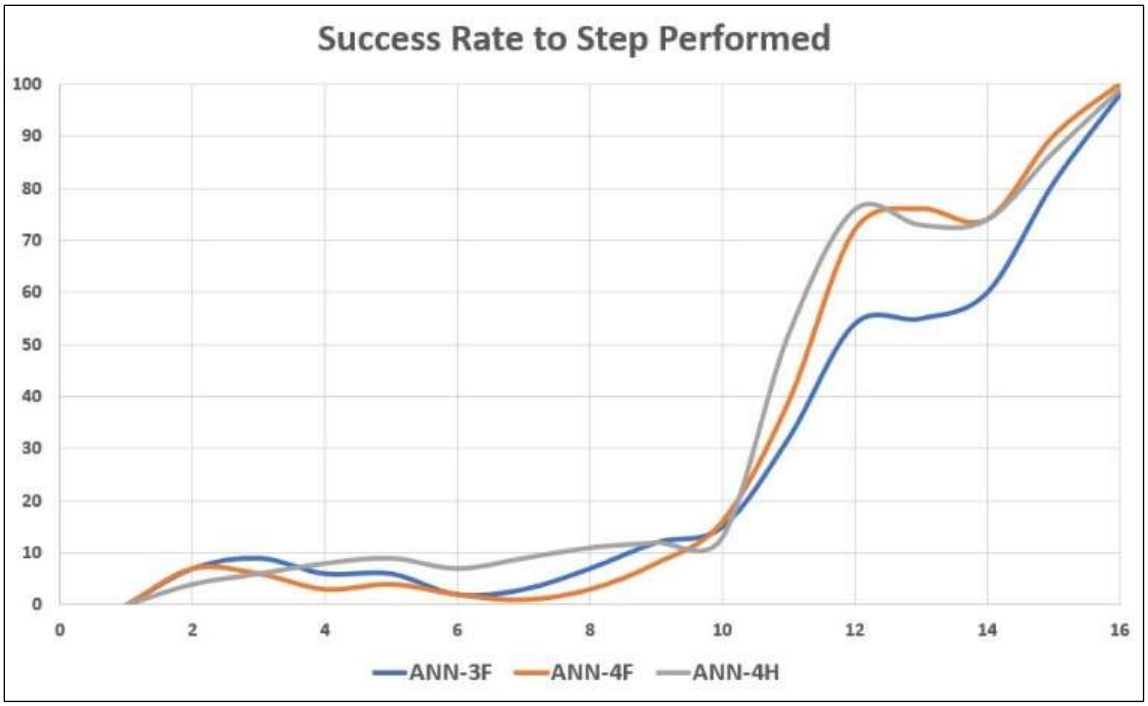

Fig.21 Success Rate Increase to Performed Step Graph for training of all 3-layer ANNs; "flat" with 3 units in hidden layer (ANN-3F), "flat" with 4 units in hidden layer (ANN-4F) and "hierarchical" for 4 units in hidden layer (ANN-4H) (Epochs: 2000)

However, both ANN4-H and ANN4-F, upon the completion of the 12th step, i.e., entering microscoping mode (looking through the ocular lenses) with the only remaining steps being those of focusing with each of the 4 objective lenses, produce a success rate of 70-80% while ANN3-F produce a success rate of approximately 55%, which is more consistent with the human expert's evaluation. That issue, combined with the fact that ANN-4F and ANN-4H show worse convergence results than ANN-3F, indicates to the possibility that in our domain, ANNs with 4 units in the hidden layer are subject to overfitting.

In all, the hierarchical reconstruction of our ANN type of *success rate* results into a more realistic assessment mechanism in terms of score increases and decreases but also into a less realistic one in terms of correspondence to the evaluations carried out by the human experts. That is the exact opposite of the hierarchical weighted average type of *success rate*, trained with a GA; there, the training of the hierarchically-redesigned weighted average resulted into greater accuracy with respect to the human evaluators but also to less explainability.

8. Conclusions and Future Work

Virtual labs are tools which can offer university students remote lab training free from the risk of accidents and damages. They also have the capacity to provide the trainees with automated evaluation of their online performance. Hellenic Open University has developed its own virtual lab, Onlabs, which fulfills both of those targets.

The evaluation of the user's performance in Onlabs is based on a scoring algorithm which has specifically been developed for this purpose. The algorithm provides the user with a two-fold real-time score; a *success rate* indicating to what extent the user has made the necessary actions for the successful completion of the experiment, and *penalty points*, which are assigned whenever the user is performing an action in the wrong order. Those two scores along with the time spent by the user on the experiment as well as the extent to which the involved instruments have been reset upon the end of the session, are combined to produce an *aggregate score* for the user's overall performance.

Initially, the parameters of the scoring algorithm are defined intuitively. In order, however, to reach more realistic parameters and subsequently a more efficient scoring algorithm, so that student evaluation in large scale can be achieved without the costly presence of human evaluators, machine learning has been used. Particularly, an interactive Genetic Algorithm and an Artificial Neural Network along with feedback from human experts are used for the calibration of *success rate*. Moreover, for those two methods, the *success rate* has also been redesigned in a hierarchical fashion, i.e., by grouping the various required actions into sub-groups according to their relevance, and the training was carried out again on this new setting. The GA and ANN training has been performed within the context of the simulated experimental procedure of microscoping.

The training results vary depending on the learning method and the setting (i.e., hierarchical or not) used. The GA gives poor convergence results in the default, non-hierarchical setting and moderately better results in the hierarchical one, while the produced *success rate* assessment mechanism is in both cases unrealistic, yet even more unrealistic in the hierarchical setting. On the contrary, the ANN most of the time converges; on top of that, the non-hierarchical ANN achieves better convergence that the hierarchical one but the latter produces a smoother and to some extent a more realistic *success rate*.

Conclusively, the calibration with machine learning of the scoring mechanism in terms of *success rate*, produces accuracy which is -at best- indicative of the student's actual performance. At the same time, the hierarchical reconstruction of the *success rate* measure trades off accuracy for realism and explainability, depending on the *success rate* type and the respective machine learning technique used. This warrants an investigation into the redesign of said hierarchy, aiming to improve both accuracy and realism, instead of just one of them.

One of our future goals is to apply machine learning training in the context of the electrophoresis experimental procedure. Since the success rate metric for the electrophoresis procedure will obviously be hierarchical, it will serve as a test-bed for a broader study of the proposed machine learning techniques in a hierarchical setting.

Finally, apart from the offline versions of Onlabs, which the user installs and plays on their computer, we have been developing an online one, which the user plays on their web browser. The latter, concerning for the moment the microscoping procedure, aims towards a much broader audience than HOU's students of biology, and indicates the need and the motive for en masse automated evaluation.

Acknowledgments

The text of this paper contains context-setting information and generic descriptions which also appear in some other work (properly referenced) by the same co-authors. The paper contains substantial new information (sections 6, 7 and 8) which constitute new work.

Funding

Acknowledgements This research has been co-financed by the Operational Program "Human Resources Development, Education and Lifelong Learning" and is co-financed by the European Union (European Social Fund) and Greek national funds.

References

Achumba, I. E. (2011). *Intelligent Performance Assessment in a Virtual Electronic Laboratory* [PhD Thesis]. University of Portsmouth.

Amir, E., & Doyle, P. (2002). Adventure Games: A Challenge for Cognitive Robotics. *American Association of Artificial Intelligence (Www.Aaai.Org)*, 8.

Ammanabrolu, P., & Riedl, M. (2019). Playing Text-Adventure Games with Graph-Based Deep Reinforcement Learning. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 3557–3565. https://doi.org/10/gf9fzz

Arora, A. (2019, February 4). *Using Genetic Algorithms to Automate the Chrome Dinosaur Game (Part 2)*. Medium. https://heartbeat.fritz.ai/using-genetic-algorithms-to-automate-the-chrome-dinosaur-game-part-2-1c0007334297

Bellotti, F., Berta, R., & Gloria, A. D. (2010). Designing Effective Serious Games: Opportunities and Challenges for Research. *International Journal of Emerging Technologies in Learning (IJET)*, *5*(SI3), 22–35. https://doi.org/10/c42vb8

Bellotti, F., Kapralos, B., Lee, K., Moreno-Ger, P., & Berta, R. (2013). Assessment in and of Serious Games: An Overview. *Advances in Human-Computer Interaction*, *2013*, e136864. https://doi.org/10/gb8hh2

Brooke, J. (1996). SUS - A Quick and Dirty Usability Scale. In P. W. Jordan, B. Thomas, I. L. McClelland, & B. A. Weerdmeester (Eds.), *Usability Evaluation in Industry* (p. 7). Taylor & Francis.

Charles, D., Fyfe, C., Livingstone, D., & McGlinchey, S. (Eds.). (2008). *Biologically Inspired Artificial Intelligence for Computer Games*. IGI Global. https://doi.org/10.4018/978-1-59140-646-4

de Freitas, S. (2008). *Serious Virtual Worlds: A Scoping Study*. Joint Informations System Committee.

Hlubocky, B., & Amir, E. (2004). *Knowledge-Gathering Agents in Adventure Games*.

Kostka, B., Kwiecien, J., Kowalski, J., & Rychlikowski, P. (2017). Text-Based Adventures of the Golovin AI Agent. *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 181–188. https://doi.org/10/gf9fz2

Luo, J. J. (2019, May 22). *An Exploration of Neural Networks Playing Video Games*. Towards Data Science. https://towardsdatascience.com/an-exploration-of-neural-networks-playing-video-games-3910dcee8e4a

Maratou, V. (2012). *Implementation of an Educational Virtual World for Software Engineering [in Greek]* [MSc Thesis]. Hellenic Open University.

Martin, M. (2011, August 30). *Using a Genetic Algorithm to Create Adaptive Enemy AI*. Gamasutra - The Art & Business of Making Games. https://www.gamasutra.com/blogs/MichaelMartin/20110830/90109/Using_a_Genetic_Algorithm_to_Create_Adaptive_Enemy_AI.php

Mendonça, V. G. de, Pozzer, C. T., & Raittz, R. T. (2008). *A Framework for Genetic Algorithms in Games*. 4.

Mislevy, R. J., Steinberg, L. S., & Almond, R. G. (2003). Focus Article: On the Structure of Educational Assessments. *Measurement: Interdisciplinary Research and Perspectives*, *1*(1), 3–62. https://doi.org/10/bf5nt9

Mislevy, R. J., Steinberg, L. S., Almond, R. G., & Lukas, J. F. (2006). Concepts, Terminology and Basic-Models of Evidence-Centered Design. In D. M. Williamson, R. J. Mislevy, & I. I. Bejar (Eds.), *Automated Scoring of Complex Tasks in Computer-Based Testing* (pp. 15–47). Routledge. https://doi.org/10.4324/9780415963572

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc.

Narasimhan, K., Kulkarni, T., & Barzilay, R. (2015). Language Understanding for Text-Based Games Using Deep Reinforcement Learning. *ArXiv:1506.08941 [Cs]*. http://arxiv.org/abs/1506.08941

Paxinou, E., Karatrantou, A., Kalles, D., Panagiotakopoulos, C., & Sgourou, A. (2018). A 3D Virtual Reality Laboratory as a Supplementary Educational Preparation Tool for a Biology Course. *European Journal of Open, Distance and E-Learning*, *21*(2). http://www.eurodl.org/?p=archives&sp=brief&year=2018&halfyear=2&article=777

Paxinou, E., Zafeiropoulos, V., Sypsas, A., Kiourt, C., & Kalles, D. (2017, June). Assessing the Impact of Virtualizing Physical Labs. *Proceedings of the European Distance and E-Learning Network 2018 Annual Conference*. European Distance and E-Learning Network 2018 Annual Conference, Genova, Italy. http://arxiv.org/abs/1711.11502

Robbins, M. S. (2019). Using Neural Networks to Control Agent Threat Response. In S. Rabin (Ed.), *Game AI Pro 360: Guide to Tactics and Strategy* (p. 242). CRC Press.

Robitzski, D. (2019, September 5). *A Neural Network Dreams Up this Text Adventure Game as You Play*. Futurism. https://futurism.com/text-adventure-game-neural-network

Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2nd ed.). Pearson Education.

Score (game). (2021). In *Wikipedia*. https://en.wikipedia.org/wiki/Score_(game)

Shute, V. (2011). Stealth Assessment in Computer-Based Gamee to Support Learning. In *Computer Games and Instruction*.

Shute, V., & Ventura, M. (2013). *Stealth Assessment: Measuring and Supporting Learning in Video Games*. The MIT Press.

Stevens, R. H., & Casillas, A. (2006). Artificial Neural Networks. In D. M. Williamson, R. J. Mislevy, & I. I. Bejar (Eds.), *Automated Scoring of Complex Tasks in Computer-Based Testing* (pp. 259–312). Routledge. https://doi.org/10.4324/9780415963572

Thomaz, A. L., & Breazeal, C. (2006). Reinforcement Learning with Human Teachers: Evidence of Feedback and Guidance with Implications for Learning Performance. *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, 1000–1005. http://dl.acm.org/citation.cfm?id=1597538.1597696

Thomaz, A. L., & Breazeal, C. (2008). Teachable Robots: Understanding Human Teaching Behavior to Build More Effective Robot Learners. *Artificial Intelligence*, *172*(6), 716–737. https://doi.org/10/d49w2c

Yourdon, E. (1989). *Modern Structured Analysis*. Yourdon Press.

Zafeiropoulos, V., & Kalles, D. (2019, October). *Human-Computer Learning Interaction in a Virtual Laboratory*. The Online, Open and Flexible Higher Education Conference (OOFHEC 2019), Madrid, Spain.

Zafeiropoulos, V., & Kalles, D. (2022). Computer-Human Mutual Training in a Virtual Laboratory Environment. In G. A. Tsihrintzis, M. Virvou, & L. C. Jain (Eds.), *Advances in Machine Learning/Deep Learning-based Technologies: Selected Papers in Honour of Professor Nikolaos G. Bourbakis – Vol. 2* (pp. 47–78). Springer International Publishing. https://doi.org/10.1007/978-3-030-76794-5_4

Zafeiropoulos, V., Kalles, D., & Sgourou, A. (2014). Adventure-Style Game-Based Learning for a Biology Lab. *2014 IEEE 14th International Conference on Advanced Learning Technologies (ICALT) 7-10 July 2014, Athens, Greece*, 665–667. https://doi.org/10/gf8x7t

Zafeiropoulos, V., Kalles, D., & Sgourou, A. (2016). Learning by Playing: Development of an Interactive Biology Lab Simulation Platform for Educational Purposes. In I. Deliyannis, P. Kostagiolas, & C. Banou (Eds.), *Experimental Multimedia Systems for Interactivity and Strategic Innovation* (pp. 204–221). https://www.igi-global.com/chapter/learning-by-playing/135131