ESTIMATING DIFFICULTY OF LEARNING ACTIVITIES IN DESIGN STAGES: A NOVEL APPLICATION OF NEUROEVOLUTION

Francisco José Gallego Durán
Estimating difficulty of learning activities in design stages: A novel application of Neuroevolution

Francisco José Gallego Durán

Submitted to get the degree of Doctor of Philosophy by Universidad de Alicante
International Doctor of Philosophy Mention

Supervised by
Faraón Llorens Largo
Acknowledgements

I would like to start this section saying that I cannot believe that this dissertation is finally finished. This dissertation is a really special work because it has years of history within. Years of ideas, experiments, tests, years of success and failures, years of excitement on seeing things work for the first time, and years of despair on finding yet another unspotted failure in experimental results. They have been many years, and that makes it look like the secret dissertation that always lived in the world of ideas but, like Plato’s, never managed to find a solid connection with real world.

Indeed, so many years of history and stories behind this dissertation makes it amazingly special. If carefully read, it can reveal pieces of its own history: a history that is definitely written in its DNA. Shy blinking bits that escaped from tens of projects have secretly populated and contributed to these pages, like magical drops. When thinking of it, I can even smell bits of all that projects that started 13 years ago with a game called Mad University. Genetic Algorithms, Neural Networks, Neuroevolution, Support Vector Machines... many things to learn, relearn,
master and start to learn again, that started to populate our projects. And all of them started with a game that humorously tried to resemble to a madhouse.

All these projects have contributed many bits of knowledge, shy and hidden bits knowledge that has finally condensed like raindrops to end up becoming part of this great project. But all these bits of knowledge, though mindfully important, would mean close to nothing if they were not part of human stories. Because projects are made of knowledge, but knowledge does not really exists without humans, and we, humans, are part of each other’s stories, like others are part of ours’.

And all these bits, bytes and projects are full of great moments due to the great persons that have been part of them, part of the stories that lead to this dissertation, part of my stories. Among these amazing persons there is my Director, in capitals, Faraón Llorens. He has been a director, a teacher, a supporter and a friend at the same time. I am more than grateful for his advise, support, patience and huge help, which has even been almost heroic many times. Also in capitals there is Rafael Molina Carmona, who has always been there to give a helping hand with a smile and whose reviews and comments have been greatly important for this work. I would also like to do a smiling reverence to my tireless mates Carlos Villagrá Arnedo, Rosana Satorre Cuerda, Patricia Compañ Rosique, Francisco José Mora Lizán, María Jesus Castel de Haro and José Requena Ruiz. Huge thanks for having helped me in so many ways across these years: being grateful to you is the bare minimum I could do. There is so many people behind this work in one or another way, that
another work would be required just to list all of them: Manuel López García, Juan Miguel Martínez Company, Miguel Ángel Lozano Ortega, Fidel Aznar Gregori, Mireia Luisa Sempere Tortosa, Mar Pujol López, Ramón Rizo Aldeguer and, of course, all the people that were part of ByteRealms, the video game registered trademark of the University of Alicante, and all the components of the Department of Computational Science and Artificial intelligence at the University of Alicante. Thanks to all of you for being part of this story, and many others.

I cannot finish this words without mentioning my parents and my brother, who have been the central supportive pillar that have taken my from the child I was to the not-so-younger child I am now. Without all their efforts, most of this stories would have never existed. I also want to have some of my deeply grateful words to my in-laws, whose kindness and support have been much more than any son-in-law could ever ask for. And last, but not least, my most special and deep thanks are for my wife, Elena, whose love and happiness are the most valuable think I have ever had. She is the most important part of this story, of all my projects, of all my bits and bytes and, the one reason of my life.

A million thanks to all of you.
Contents

Acknowledgements

List of Figures

List of Tables

Abstract

1 Introduction

2 Background
3 Measuring Difficulty of Learning Activities 41
  3.1 Sources for measuring difficulty 43
  3.2 Desired properties for difficulty 45
  3.3 Intrinsic limitations 48
  3.4 Mathematically defining difficulty 51
  3.5 Improving the definition of difficulty 52

4 PLMan: A Game-based Learning Activity 57
  4.1 PLMan: The game 58
  4.2 Developing controllers for mazes 62
  4.3 Examples of developed controllers 65
  4.4 PLMan as learning activity 67
  4.5 Measuring difficulty of PLMan mazes 70

5 Estimating Learning Difficulty with Neuroevolution 77
  5.1 How to estimate difficulty 78
  5.2 Learning to play PLMan automatically 80
  5.3 Designing the neural networks 81
  5.4 Measuring learning difficulty for Neuroevolution 87
  5.5 Similarities between humans and Neuroevolution 91

6 Results and Discussion 95
  6.1 Environment setup for Neuroevolution 96
  6.2 Fine tuning NEAT and HyperNEAT 99
  6.3 Experimental results 105
  6.4 Discussion on global results 111
List of Figures

2.1 An example of a NEAT genome. Link genes are historically tracked using a unique InnovationID among epochs. Also, when new neurons are added, previous links do not disappear: they become disabled. That lets the crossover operator restore its previous status depending on the other mate being crossed over. ........................................... 29

2.2 Example of a crossover between two NEAT genomes. Right genome has a higher fitness value than left genome. Genes marked in red are those inherited by the descendent. ... 30

2.3 2D neuron substrate with 16 nodes (left) and a generic CPPN used to produce weight values for connections between neurons (right). ............................... 32

3.1 Representation of the flow channel, by Csikszentmihalyi (1990) ................................................................. 42
3.2 Manually constructed difficulty function for basket example. Difficulty decreases as player progresses, scoring baskets in this example. ........................................ 48

3.3 Behaviour of $D_t$ using equation 3.4 with data from the basketball example from section 3.2. Left, exact definition for $E_t$ with step value changes. Right, linear interpolation for $E_t$. ............................................. 55

4.1 Example maze along with the Prolog knowledge base that controls Mr.PLMan (@) to eat all the dots dodging the enemy (E) ............................................. 59

4.2 PLMan game's turn-based execution cycle .................................. 60

4.3 PLMan's short-range visual sensor. Mr.PLMan (@) is able to see two walls (#), an enemy (E), a dot (.), an object (a) and some empty spaces ( ) ............................................. 62

4.4 PLMan's long-range visual sensor ............................................. 63

4.5 An example of a solved maze from PLMan's stage 1. .... 66

4.6 An example of a solved maze from PLMan's stage 2. .... 68

4.7 $E_t$ and $D_t$ measured for PLMan maze 1-00 (up) and 1-09 (down). ............................................. 72

4.8 $E_t$ and $D_t$ measured for PLMan maze 2-13 (up) and 2-32 (down). ............................................. 74
5.1 On the left, 4 examples of Atari games that Neuroevolution learns to play. On the right, an Atari joystick along with its 9-command representation (8 movement commands and 1 fire button).

5.2 Input design using PLMan’s short range sensor for NEAT

5.3 NEAT/HyperNEAT output design with explicit actions

5.4 NEAT/HyperNEAT output design split in actions and movements

5.5 HyperNEAT input design. There is an input substrate for each type of object that can be found on the maze. The size of each input substrate is the same as the selected input grid. Selected grid is 3x3 in this example.

5.6 PLMan maze 2-45, classified as stage 2, difficulty 2 by professors.

5.7 Overall design of the proposed HyperNEAT architecture. 2 fully-connected, recursive, hidden substrate layers are the core. Input substrates are based on Mr.PLMan’s short range sensor. Output substrate contains two neuron blocks with 4 output neurons each (for selecting action and direction).

5.8 Graphically measuring similarity between $D_t$ and $\tilde{D}_t$ for a given maze $z$. Area between the curves represents their distance (dissimilarity). The white area of the graph represents their similarity (complement to their distance).
6.1 Implementation of NEAT/HyperNEAT neural network controllers for PLMan ................................. 97

6.2 Training model for NEAT/HyperNEAT controllers ................................................................. 98

6.3 Mazes selected to be used for finding the best set of parameters. These mazes include enemies (E), follower phantoms (F), keys (a), doors (\), levers (\_), pushable blocks (\%), automatic archers (V), laser shots (\_), bombs (+) and guns (I). ........................................... 103

6.4 Difficulty and similarity results for NEAT and HyperNEAT on maze 1-41 (Stage 1, difficulty 3). $D_t$ is the difficulty function measured for students that were assigned this maze. $E_t'$, $D_t'$ and $E_t''$, $D_t''$ are the easiness and difficulty functions measured for NEAT and HyperNEAT. Distance is the graphical representation of $1 - \sigma_t^A$, for $A \in \{\text{NEAT, HyperNEAT}\}$. ......................................................... 108

6.5 Difficulty and similarity results for NEAT and HyperNEAT on maze 2-11 (Stage 2, difficulty 5). $D_t$ is the difficulty function measured for students that were assigned this maze. $E_t'$, $D_t'$ and $E_t''$, $D_t''$ are the easiness and difficulty functions measured for NEAT and HyperNEAT. Distance is the graphical representation of $1 - \sigma_t^A$, for $A \in \{\text{NEAT, HyperNEAT}\}$. ......................................................... 109

6.6 Distribution of similarity for NEAT and HyperNEAT ............................................................ 113
6.7 Mazes 2-06 (left) and 2-31 (right). Neither NEAT nor HyperNEAT found a controller able to eat a single dot after 2000 iterations.

A.1 Elements of the PLMan Learning System

A.2 Students’ view of the gamified Website at a given instant.

A.3 Evolutive probabilistic graph for a great student. Green color represents the probability of achieving great marks, blue the probability of normal marks, and red the probability of underperforming marks.

C.1 Difficulty and similarity results for NEAT and HyperNEAT on maze 1-31 (Stage 1, difficulty 1).

C.2 Difficulty and similarity results for NEAT and HyperNEAT on maze 1-08 (Stage 1, difficulty 2).

C.3 Difficulty and similarity results for NEAT and HyperNEAT on maze 1-56 (Stage 1, difficulty 4).

C.4 Difficulty and similarity results for NEAT and HyperNEAT on maze 1-59 (Stage 1, difficulty 5).

C.5 Difficulty and similarity results for NEAT and HyperNEAT on maze 2-40 (Stage 2, difficulty 1).

C.6 Difficulty and similarity results for NEAT and HyperNEAT on maze 2-48 (Stage 2, difficulty 2).

C.7 Difficulty and similarity results for NEAT and HyperNEAT on maze 2-49 (Stage 2, difficulty 3).
C.8 Difficulty and similarity results for NEAT and HyperNEAT
on maze 2-56 (Stage 2, difficulty 4). . . . . . . . . . . . 159
List of Tables

6.1 Candidate parameter sets tested using the proposed greedy approach .................................................. 102
6.2 Selected parameters for NEAT and HyperNEAT after testing the set of candidate parameters using the proposed greedy approach ................................................................. 105
A.1 Structure of stages for mazes to be solved by students in the automated system ............................................ 129
C.1 Complete results for mazes tested in this research: mazes 1-00 to 1-29 ......................................................... 144
C.2 Complete results for mazes tested in this research: mazes 1-30 to 1-59 ......................................................... 145
C.3 Complete results for mazes tested in this research: mazes 2-00 to 2-29 ......................................................... 146
C.4 Complete results for mazes tested in this research: mazes 2-30 to 2-59 ......................................................... 147
Abstract

In every learning or training environment, learning activities are the basis for practical learning. Learners need to practice in order to acquire new abilities and perfect those previously gained. The key for an optimized learning process is correctly assigning learning activities to learners. Each learner has specific needs depending on previous knowledge and personal skills. A correct assignment for a given learner would be selecting a learning activity that closely matches learner’s skills and knowledge. This brings up the concept of difficulty. Difficulty of a learning activity could be defined as the effort that a learner has to make to successfully complete the learning activity and obtain its associated learning outcomes. So, a difficult activity would simply require much effort to be successfully completed.

Learners presented with too difficult learning activities tend to abandon rather than performing required effort. This situation could be better understood as the learner perceiving the activity as an unbalanced invested-return ratio: too much effort for the expected learning outcomes. A similar case occurs when difficulty is too easy. In that case, effort per-
ceived is low, but learning outcomes are perceived as even lower. If the activity does not pose a challenge for the learner is because the learner already masters the involved abilities, and that makes learning outcomes tend to zero. Both situations drive learners to losing interest.

To prevent this from happening, teachers and trainers estimate difficulties of learning activities based on their own experience. However, this procedure suffers an effect called the Curse of Knowledge: every person that masters an activity, becomes biased for estimating the effort required to master that same activity. Therefore, correctly estimating difficulties of learning activities is an error-prone task when expert-knowledge is used to estimate them. But estimating difficulty without carrying out the learning activity would probably yield even worse results.

In order to escape from this error-prone cycle, the first solution would be to measure the effort involved in successfully completing the learning activity. For that purpose, an objective effort measurement should be defined. This approach has been followed by many previous works and it is the general approach in the field of Learning Analytics. Although this approach yields many types of considerable results, it has an important drawback. It is impossible to have a measure without learners performing the learning activity. Therefore, at design stages of the learning activity, how does the designer know whether the activity is too hard/too easy? Is there a way to have an valid estimation of difficulty of a learning activity before handing it to learners?

This work proposes a new approach to tackle this problem. The approach consists in training a Machine Learning algorithm and measure
the “effort” the algorithm requires to find successful solutions to learning activities. The “effort” will be the learning cost: the time the algorithm requires for training. After that, results obtained from training the Machine Learning algorithm will be compared to results measured from actual learners. Under the assumption that learning costs for Machine Learning algorithms and those for learners have some kind of correlation, results from comparing them should show that correlation. If that were the case, then the learning cost that Machine Learning algorithms invest in training could be used as an estimation of the difficulty of the learning activity for learners.

In order to implement this approach and to obtain experimental data, two Neuroevolution algorithms have been selected for the Machine Learning part: Neuroevolution of Augmenting Topologies (NEAT) and Hypercube-based Neuroevolution of Augmenting Topologies (HyperNEAT).

Implementing this proposed approach has yielded several contributions that are presented in this work:

- A new definition of difficulty as a function, based on the progress made over time as an inverse measure of the effort/learning cost.

- A similarity measure to compare Machine Learning results to those of learners and know the accuracy of the estimation.

- A game called PLMan that is used as learning activity in the experiments. It is a Pacman-like game composed of up to 220 different mazes, that is used to teach Prolog programming, Logics and a light introduction to Artificial Intelligence.
• An application of NEAT and HyperNEAT to learn to automatically solve PLMan mazes.

• A novel application of Neuroevolution to estimate difficulty of learning activities at design stages.

Experimental results confirm that there exists a correlation between learning costs of Neuroevolution and those of students. Goodness of the presented results is limited by the scope of this study and its empirical nature. Nevertheless, they are greatly significant and may open up a new line of research on the relation between Machine Learning and humans with respect to the process of learning itself.
Chapter 1

Introduction

Learning is a fascinating event that happens in nature as the maximum expression of adaptation. Having the ability to learn means being able to remember past events and associate them with present situations to take decisions. Most animals show this ability with different degrees of performance. It intuitively seems to be a key feature for any creature to subsist. That would be, according to Darwin (1859), a good reason to explain the widespread of Intelligence across most of the creatures in the Earth.

Interest in how Learning happens have been present in human culture since first known documented studies. Although research has achieved many interesting results related to Learning itself, it is still mostly unknown how it actually happens. This leaves training and education in the realm of practical knowledge, still far away from having a complete low-level scientific understanding of how and why learning happens. There-
fore, much research is required in two main lines: 1) empirical studies that gather more information about indirect connections between activity and learning, and 2) theoretical research to find better high-level explanations of this phenomena.

Life experience suggests that there is a strong correlation between the activities any individual performs and Learning. It makes sense that having more experiences provides more information and improves the ability of remembering and relating them to future situations. This is the basis for training and education: creating experiences to make learners’ brains adapt and remember. In fact, learning experiences designed by some teacher/trainer in order to generate some concrete learning outcomes on performers of the experience are often referred to as Learning Activities.

The space of possible learning experiences to create is potentially infinite. How to know which experiences will be useful for future situations? Assuming two experiences with same learning outcome, which one is better? Why? Is there a way to search for optimal learning experiences? These questions drive the academic research on Learning. Under these questions, and assuming there is still no theoretical explanation on the exact way Learning happens, there is some consensus on several practical lessons:

- Practitioners assume that Learning depends on previous status of learners’ brains (i.e. previous knowledge) (Ley and Kump, 2013, Prensky, 2001, Redecker et al., 2012). This implies that optimal learning experiences, assuming they exist, will be different for each learner.
• Learners’ motivation seems to play a key role in effectiveness of the learning experiences. Experience seems to state that motivated learners achieve better learning outcomes faster (Prensky, 2001).

• Motivation seems to be affected by the sense of progress (Cocea and Weibelzahl, 2006, Koster and Wright, 2004, Wang and Newlin, 2000). Learners feeling that their effort pays off in terms of progression seem to increase their motivation and be willing to invest more effort in learning activities.

• Ultimately, sense of progress is affected by the nature of the learning activities performed (Hu et al., 2014, Koster and Wright, 2004). When they are well suited for the learner, motivation takes place, activities get done and the learner gets learning outcomes.

• A learning activity is considered as well suited for a learner when it matches present learner’s knowledge and capabilities, and connects them with a new knowledge or a new development level with respect to some ability. This could be visualized as a step, matching next and previous steps on a stair, or as a link, matching next and previous links on a chain.

Considering the concept of well suited learning activities for a learner, the intuitive notion of difficulty appears. Difficulty could be defined as the cost\(^1\) the learner has to pay in order to successfully realize the learning activity and acquire its learning outcomes. This is also connected to

\(^1\)Cost here should be interpreted as amount of effort: a combination of dedication time and concentration that are required.
the previous knowledge the learner requires: if an activity requires some knowledge the learner does not posses, this knowledge will have to be acquired to be able to perform the activity, increasing the cost.

An activity is considered difficult when it requires a great cost to be successfully realized. Conversely, an easy activity would be one involving little effort. Although these considerations tend to be subjective and dependent on each learner, this concept is universally known and used. Moreover, this concept has implications in all previously enumerated practical lessons:

- A too difficult activity requires too much effort from the point of view of the learner. This means that the learner may perceive the activity as having a poor investment-return ratio. It could also mean that the activity requires the effort to be performed within a very restricted time-frame, which could be impossible or almost impossible for the learner.

- On the contrary, a too easy activity would require almost no effort from the learner. This usually means that the learner already possesses the learning outcomes the activity produces. If the activity requires some considerable amount of time to be carried out, it could also be considered as having a poor investment-return ratio: investing some time to obtain almost no learning outcome does not seem a good idea.

As a consequence, a well suited activity for a given learner should be difficult enough: neither too difficult, nor too easy. It is widely consigned
that difficult enough activities have the potential to motivate learners. By definition, these activities accurately match learner’s abilities, so having a perfect investment-return ratio. This may be a valid explanation on way these activities have such motivational potential: they are perceived as an optimal way to acquire knowledge or abilities. Moreover, the feedback produced when realizing these range of learning activities also contributes to a positive sense of progress.

The problem arises when measuring and/or estimating difficulty for any given learning activity (Aponte et al. (2009), Missura and Gartner (2011), Mladenov and Missura (2010), Mourato and dos Santos (2010), Ravi and S. (2013)). As discussed before, the universal concept of difficulty is fully subjective and usually defined by comparison between different learning activities and learners. It is generally easy to find different performers for any given learning activity that report different values when asked to measure its difficulty. Clearly, measuring and estimating difficulty is a subjective task: the usual way to estimate the difficulty of any learning activity is by guesswork based on guesser experience. Teachers and trainers estimate difficulty of learning activities based on their experience on the subject being learnt. However, using experience as a source for measuring difficulty of any learning activity is a task affected by the Curse of Knowledge (Colin Camerer, 1989). Once a person masters a learning activity, difficulty for mastering it disappears. Therefore, when the person reflects on the difficulty of any already mastered learning activity, any estimation will tend to be biassed by the acquired knowledge (generally, a underestimation bias). Not surprisingly, these estimations
One way of addressing this is by measuring statistical indicators out of learners’ historical results (Cheng et al., 2008, Lykourentzou et al., 2009, Ravi and S., 2013, Romero et al., 2013). Based on results, many interesting measures can be obtained: required time to realize an activity, success percentages, number of failed attempts before succeeding... All these measures depend on the activity itself, but the point is getting enough factual data to derive comparative indicators between activities. This has been done many times with different degrees of success and utility (Cheng et al. (2008), Hu et al. (2014), Lykourentzou et al. (2009), Park and Kanehisa (2003), Romero et al. (2013), Zafra et al. (2011)). As previous works suggest, this approach is very promising as it is the basis for a complete new field called Learning Analytics (Siemens (2012)).

Learning Analytics (Siemens (2012)) often focusses on analysing learners and learning activities through applying statistics and Machine Learning to gathered data. Getting a well defined corpus of data about a learning activity enables better understanding of the activity itself, its learning outcomes and its difficulty. This is a great advance, but with a cost: some learners have to be “sacrificed” to get enough data. When a new activity is created, there is no empirical evidence about its difficulty and/or learning outcomes. Because of this, an initial estimated value of difficulty tends to be manually assigned by experts based on their experience. As this is an error-prone practice, first students confronting new activities will have to pay the additional cost of erroneous difficulty estimations.
This leads to a critical question: is there another way to obtain estimations about difficulty of learning activities at design stages? If it does exist, could it be automated? Could it be optimized and improved? Automatically producing estimations would represent an interesting contribution: it would relieve teachers and trainers’ time required to perform this task. This freed time would then be available for other more important tasks like designing new activities, for instance. Moreover, automating estimations is only a first step: once they are automated, researchers can deepen in the knowledge about how learning actually works. This may lead to new discoveries and create a feedback loop that might ultimately transform the whole teaching-learning system.

This research starts by hypothesizing that there are ways to define difficulty able for being automatically measured and estimated. In this context, estimating refers to the ability to predict the final difficulty value before collecting actual data about any given learning activity\(^2\). All the work carried out in this research is made on the basis that this is possible, and the aim is to obtain a first empirical validation of this hypothesis. Validating that this may be possible in practice, even if proves are bound to some specific data, is a first step to motivate and invite some more research on this field. Driving attention to this unexplored possibility may lead to researchers producing more practical experiences and results and, eventually, developing a working theoretical corpus.

\(^2\)Predictions or estimations for a given learning activity should be possible to be performed during design stages of the activity.
1.1 Goals and hypothesis

Let us start by assuming the previously stated hypothesis that there exist definitions for difficulty that let us automate measurement and estimation. Then, the goal of this work is designing and performing experiments to prove this hypothesis empirically. Validation will be limited to the proposed context, but that will be considered enough for this work, as it is a solid starting point for further research.

To achieve the final goal of validating or dismissing the proposed hypothesis, this plan will be followed:

- Create a definition of difficulty of a learning activity that can be used to measure the learning cost of a general learner. A general learner would be any agent that performs learning activities and improves its results. This improvement is a consequence of learning, be the subject a human or a Machine Learning algorithm.

- Define general indicators and functions to measure difficulty, and derive some particular implementations for specific problems and proposed datasets. For this task, it will be important to concretely design desired properties that indicators and functions should accomplish. The main focus is to create procedures to automatically measure and estimate difficulty.

- Select a particular context and apply Machine Learning to estimate difficulty of a specific learning activity. Produce adaptations of the functions and indicators required for the selected activity and for the Machine Learning algorithms that will be used to generate
1.1. Goals and hypothesis

- Design indicators to measure the accuracy of the estimations generated using Machine Learning on previous step. Estimations should be considered better as they become closer to actual measured data. Therefore, indicators will have to measure similarity between estimations and measures from actual collected data.

- Finally, analyse similarity measures and search for correlations between estimations and actual collected data. The existence of correlations would validate the hypothesis for the specific data used in this work.

All these goals are based on the main hypothesis of this work, which can be outlined as follows:

*The training stage of a Machine Learning algorithm can be interpreted as the cost that the algorithm has to pay to learn. Humans also have to pay a cost to learn, in the form of invested time and effort. Are these two costs related in any sense? Is it possible to find correlations between them? If correlations do exist, are they strong enough to accurately estimate learning costs for humans out of those for programs?*

This work assumes that the answer to these questions is yes, and aims to empirically prove it in a proposed practical context.
1.2 Motivation

Over the years, one of my personal interests have been optimizing the learning process. Every time I have had the opportunity to assist as student to any kind of lessons, I have ended up analysing them. Some intuitive and thoughtful ideas have resulted from these analyses: things like what makes lessons boring, the importance of motivation, the value of effort or what kind of activities are more productive for obtaining knowledge and/or abilities. There are too many aspects related to learning, and that makes finding optimal ways of learning a greatly challenging task. Moreover, any achieved improvement, however small, comes with great rewards: helping students and learners in general to achieve their goals is a highly gratifying aim.

Transmitting knowledge or promoting learners to acquire abilities in the best possible way is extremely challenging. The greatest problem is the lack of solid scientific understanding about the actual processes that ultimately lead to learning. All of us have a sense of what learning is and how to achieve learning goals, but we also perceive that learning seems to be greatly dependent on each learner. Under the present lack of factual knowledge about how our brains exactly learn, optimising knowledge transfer to a great amount of different learners is even harder.

Due to my personal experience and knowledge, the key factors I consider for optimizing learning are motivation and difficulty. From my personal point of view, a learner with maximal motivation towards a perfectly balanced learning activity (with respect to its difficulty) is the optimal setup for learning. This view is what has driven this work to its
present status. I do not know exactly if this view is completely certain, partially certain or misleading, but I am grateful to it for being the root of the results presented here.

1.3 Main contribution and expected impact of this work

This work proposes a new way of defining difficulty of learning activities in order to be able to measure it and to produce predictions and estimations of its value. All the work is based on the idea that Machine Learning and human learning may be correlated in some sense. Understanding difficulty as a cost for achieving learning outcomes, this work proposes that the cost Machine Learning has to pay for learning may be correlated with the cost humans have to pay.

The main contribution of this work is proving that both proposed costs are correlated, at least for a given specific dataset. As proposed dataset comes from actual learning activities being currently carried out at the University of Alicante, the existence of a correlation is to be considered an initially significant event. This is so because such a dataset could be considered a random sample from the distribution of all possible datasets coming from actual learning scenarios. If there were no correlation between Machine Learning and humans on their learning costs, there would be a high probability for any sample of showing no correlation. Therefore, assuming that the proposed sample has no specific bias, proving the existence of a correlation in it is a significant first step
towards using Machine Learning as predictor for the difficulty of newly created learning activities.

Findings in this research suggest that there is a correlation between Humans and Machine Learning in their learning costs. As this enables the use of Machine Learning as a predictor for the difficulty of learning activities at design stages, expected impact of this work includes (but it is not limited to):

- Promoting research in a new line about the relation between Machine Learning and human learning with respect to difficulty.
- Encouraging the design of learning activities in a measurable way compatible with analyses and estimations.
- Helping to improve the knowledge about how learning actually happens.
- Helping to find new ways to optimize the learning process through adequately estimating difficulty of learning activities, enabling to automatically match learners with most appropriate learning activities.
- Helping to automatic customization of the learning process, providing new tools for existing Intelligent Tutoring Systems.

1.4 Approach

To achieve a successful result in realizing this work, a methodological approach has been carried out. This approach is summed up in this
1.4. **Approach**

essay as follows:

- This first chapter provides an introduction to this work, presenting the reasons that motivated its realization together with the main hypothesis and goals.

- Chapter 2 reviews the current state of the art with respect to the main subjects on which this work is supported: Neuroevolution, Learning Analytics and student performance prediction.

- Chapter 3 introduces the proposed definition of difficulty as a function on the effort over time. This function is the basis in which relies all presented work, because all measures of difficulty of learning activities are taken using this definition.

- Chapter 4 presents the PLMan game. PLMan is a game currently being used at the University of Alicante to teach Prolog, Logics and a light introduction to Artificial Intelligence. Problems that this game presents to students are learning activities that meet the requirements to be measured and analysed using the proposed definition of difficulty. All experimental data used in this work come from PLMan.

- Chapter 5 explains the method followed to use Neuroevolution as predictor of the difficulty of learning activities. Although arguments are presented in a general way, adaptations and formulae are developed for PLMan, as it is required for the experiments. A similarity function is also presented to measure the accuracy of predictions.
• Chapter 6 shows the experiments carried out and the final results. It also provides details on the way experiments were designed and implemented, and a discussion on the significance of the final results achieved.

• Chapter 7 concludes summarizing all the contributions presented in this work and showing the future research lines considered.

• Appendix A contains complete details of the learning system used in the University of Alicante along with PLMan. This system automatically presents students with their assignments and assesses their results.

• Appendix B shows a detailed explanation on the configuration parameters for the selected Neuroevolution algorithms.

• Appendix C holds complete result tables and graphs on the experiments carried out, along with detailed explanations and interpretations on their significance.

• Appendix D is a summed up version of this complete work in Spanish. It includes abstract, introduction, results and conclusions.
Chapter 2

Background

The contributions presented in this work are founded in very different fields of knowledge, as this work has a strong interdisciplinary nature. The roots of this work are on modern learning methodologies coming from the field of Learning Analytics (Siemens, 2012): student performance data is collected and analysed automatically. This analysis is key for dynamic adaptation of automatic learning systems. This work is focused on the concept of difficulty of learning activities, generating automatic measures and predictions that should be further used to make learning systems evolve.

With respect to the Analytics part and the predictive part in the goals of this work, the field of Machine Learning is completely crucial. The kind of predictions that are required for this work are related to the concept of Reinforcement Learning in its broader sense: the task of automatically learning to solve learning activities, usually requires performance mea-
sures or fitness functions to direct the learning process towards its goal. For this research, Neuroevolution algorithms have been selected. The potential of Neural Networks to adapt to problems with many inputs and outputs, like controlling an agent in an environment, is a key feature for the purposes of this work.

Moreover, previous works dealing with the concept of difficulty and the ability to generate predictions are also important for this research. These works also tend to be closely related to the fields of Computer Games and Gamification. However, it is interesting to take into account that Computer Games are learning activities by themselves. Because a computer game is an almost pure intellectual activity, it has a direct connection to learning. Taking into account words from Koster and Wright (2004): “That is what games are, in the end. Teachers. Fun is just another word for learning.” Therefore, works related to difficulty in Computer Games are also works on learning itself.

2.1 Learning Analytics and Predictive Systems

Data generated by technological systems is growing exponentially over the years. Decisions based on evidence require conclusions to be extracted from data. Researchers and/or practitioners are unable to process all this information in a short period of time. However, decisions cannot wait for data to be processed in most environments. On the verge of this growing importance of data-driven decision-making, many new disciplines
are emerging: Business Intelligence, Health Care Analytics, Social Media Analytics, and so on (Long and Siemens, 2011).

The recently named field of knowledge devoted to study the learning process out of data is Learning Analytics (Kotsiantis, 2012, Siemens, 2012, Tinto, 1987), closely related to its sibling field Educational Data Mining (Romero and Ventura, 2007). According to the 1st International Conference on Learning Analytics and Knowledge (LAK, 2011), Learning Analytics is the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimizing learning and the environments in which it occurs. This description matches precisely with most of the intentions of this work: in order to optimize the learning process, learning activities have to be adequately selected and presented to learners (Khribi et al., 2008, Vozniuk et al., 2013). In fact, many studies stress the importance of learners being presented with activities that closely match their skill level (Aponte et al., 2009, Herbrich et al., 2007, Missura and Gartner, 2011, Mladenov and Missura, 2010, Mourato and dos Santos, 2010). In other words, difficulty of the activities should be neither too high nor too low (Koster and Wright, 2004).

Optimizing the learning process is the commonly shared goal of all researchers and practitioners in these fields. Improving the performance of students in any given learning system concentrates all research efforts, either directly or indirectly. Many different approaches have emerged pursuing this holy grail: some recognize the importance of self-assessment (Roll et al., 2011, Verdú et al., 2012), others focus on Intelligent Tutor-
ing Systems (D’Mello et al., 2012, Huang et al., 2006, Roll et al., 2011, San Pedro et al., 2013), or do automatic assessment (Villagrá-Arnedo et al., 2009, Vujošević-Janičić et al., 2013, Wang et al., 2011), or try to predict student or group performance (Illanas Vila et al., 2013, Ley and Kump, 2013, Petkovic et al., 2012, Schalk et al., 2011, Yoo and Kim, 2014), or even combine this with games or Gamification (Illanas Vila et al., 2013, Iosifidis, 2011, Villagrá-Arnedo et al., 2009). All this proposals are based on analysing student data, in one or several different ways, to obtain evidence about the learning process.

Obtaining usage data from a Learning Management System (LMS) (Bíró, 2014, Lewis et al., 2005, Watson and Watson, 2007) is relatively easy today. LMSs store usage statistics in a well-structured fashion and they usually let administrators exporting this data, or creating plug-ins for this task. However, learning itself is an extremely complex process, far from being understood by science. This inherent complexity cannot be simplified with present tools and/or knowledge. Because of this, Long and Siemens (2011) consider that there is a risk to return back to behaviourism if we limit Learning Analytics to behavioural data. The current challenge of Learning Analytics is escaping from behavioural data and/or simple usage statistics, and obtaining relevant information about attitudes, skills and learning results. So, the challenge is extracting evidence about the learning process itself.

One of the most interesting uses of data is feeding predictive systems (Dekker et al., 2009, Hu et al., 2014, Huang and Fang, 2013, Ley and Kump, 2013, Lykourentzou et al., 2009, Nghe et al., 2007, Thai-Nghe
et al., 2009, Yoo and Kim, 2014). These systems use different Machine Learning techniques to discover relations between behavioural data produced by student interaction with an LMS and some concrete events like student drop out, probability of success or even student final marks. Some researchers argue that Learning Analytics can help identify at-risk learners and provide intervention to assist learners in achieving success (Macfadyen and Dawson, 2010). Kotsiantis (2012) made an interesting review about Learning Analytics and prediction. Practitioners tend to build a model once in time and use it to predict future student performance in form of grades or marks. An interesting example is found in (Huang and Fang, 2013), where authors compare four Machine Learning models for predicting student performance. Notably, their research is focused on the quality of predictions for a concrete moment in time, and they do not consider data aggregation or evolution in time. Also, Lykourentzou et al. (2009) achieve an accurate prediction at an early stage of an e-learning course using feed-forward neural networks. However, the proposed method failed to predict precisely the performance of certain specific students.

Some authors have used learning indicators such as final scores from the Standard Admission Tests (SAT). Others prefer using behavioural Learning Management System (LMS) data like user logins. For instance, Schalk et al. (2011) built a Machine-Learning-based predictive system to determine which students were at risk of failing introductory courses in mathematics and physics. The system used Random Forests technique to model SAT results data from previous years. While their results
are good, the method designed was not thought to be maintained over time, nor to do progressive predictions based on incremental information. This is a constant on previous research found in literature: the majority of the proposed methods focus on the accuracy of the Machine Learning system, but let apart factors like temporal coherence, variations of data in time or even reusability of the system. In (Wang and Newlin, 2000, 2002), the authors proposed the use of data from a web-based LMS about student on-line activity and provided an early indicator of student academic performance. Other works (Macfadyen and Dawson, 2010) conclude that student tracking data obtained from LMS may provide pedagogically meaningful information about engagement and likelihood of success. Several authors have concluded that LMS usage patterns and student performance are related (Campbell and Oblinger, 2007, Goldstein and Katz, 2005), and the combination of SAT scores and LMS logins have a considerable predictive potential. Their results show that most accurate predictions are produced when combining both sources of information: SAT scores and behavioural data.

2.2 Machine Learning and Neuroevolution

During the last twenty years, Machine Learning (Mitchell, 1997, Vapnik, 1995, 1998) has grown exponentially to become one of the most relevant fields in Computer Science. The great amount of available computational power along with the global dissemination of the Internet have been two key factors that have contributed to spreading the great work
Great advances have been undertaken and impressive results obtained. As Fernández-Delgado et al. (2014) shows in his review, hundreds of practical and theoretical neuroscience research projects have been conducted, raising our understanding of the human brain and its hidden learning mechanisms. And Fernández-Delgado et al. (2014) only considers supervised classifiers. However, Machine Learning is still far away from achieving the most ancient goal in the field: a way to compute information that can compete with the superb ability of our brains to learn, classify, analyse and archive information. Core algorithms in Machine Learning are generally supervised classifiers, because those are the kind of algorithms that are easily applied to real world problems with a considerable success ratio. However, these algorithms are not suitable for more human-like activities, in the way humans do. Humans do not start with huge tables of information and a clear model function to optimize. Humans learn by experience: they obtain the information from the environment at the same time that they are interacting with it. In this sense, supervised classifiers are unable to learn by themselves to perform activities such as playing soccer or to identifying a person out of his way of walking. They require expert tuning and a great amount of previously processed data from the field.
There is a subclass of Machine Learning algorithms designed for learning from direct experience with a specific activity. This subclass of algorithms is called Reinforcement Learning methods (Gosavi, 2009, Kaelbling et al., 1996). What this algorithms do is training agents by directly making them interact with the environment where they are supposed to perform a given activity. Then, iteration after iteration, the algorithm evaluates the specific actions performed by the agent and issues “rewards”. This “rewards” are generally bonus points in a defined evaluation metric: approximately the same as a fitness function being partially evaluated at distinct time steps. This method approximately mimics the way humans learn: by receiving feedback from the environment that indirectly informs about the success or failure of performed actions. This type of Machine Learning algorithms are better suited to deal with problems related to control systems, robotics, game playing or simulations (Galway et al., 2008, Kormushev et al., 2013, Koutník et al., 2014, Lucas, 2005, Tziortziotis et al., 2014).

2.2.1 Neuroevolution

Following a completely similar approach to Reinforcement Learning methods, the field of Neuroevolution deals with a combination of Artificial Neural Networks and Evolutionary Algorithms to train agents that learn directly from experience (Eiben and Smith, 2003, Kowaliw et al., 2014, Schaffer et al., 1992, Stanley, 2004, Stanley and Miikkulainen, 2002d, Wagner and Altenberg, 1996, Whitley et al., 1990, Yao, 1999). Evolutionary Algorithms are used to automatically generate and optimize network
topologies together with their weights. This process produces trained Artificial Neural Networks and is usually referred to as “evolving networks”. This is usually done by starting with randomly generated populations of Artificial Neural Networks. These networks are tested against a given environment, task or problem, measuring their fitness. The fitness is used to select best adapted individuals and to cross them over to produce new populations. Some methods rely in a crossover operator, others only on a mutation operator, and most advanced methods have both operators and let the user combine them at will. Neuroevolution algorithms are particularly suitable for the same tasks as Reinforcement Learning: they deal with problems where the correct output is not possible to establish beforehand for each set of inputs.

This field started more than 25 years ago by combining Artificial Neural Networks and Genetic Algorithms (Schaffer et al., 1992). Tens of algorithms have been conceived and developed following this combination pattern (Gruau et al., 1996, Opitz and Shavlik, 1997, Yao, 1999, Zhang and Mühlenbein, 1993) leaded the first interesting insights in the field of Neuroevolution. Algorithms like Generalized Acquisition of Recurrent Links (GNARL) (Angeline et al., 1994), Symbiotic Adaptive Neuro-Evolution (SANE) (Moriarty, 1997), Enforced Subpopulations (ESP) (Gomez and Miikkulainen, 1997, 1999), GasNets (Husbands et al., 1998) yielded the first limited but most promising results. However, Neuroevolution has become mainstream on past 10 years with the growth of new successful algorithms like EANT (Kassahun et al., 2009), and particularly Neuroevolution of Augmenting Topologies (NEAT) (Stanley, 2004)
and Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) (D’Ambrosio, 2011a). The success of these latest algorithms is based on their ability to deal with the Competing Conventions Problem (also known as Permutations Problem (Radcliffe, 1993)), in the case of EANT and NEAT, and on the capability to produce large-scale neural networks that can profit from geometric layout of neurons though patterns of symmetries and repetitions. With respect to EANT and NEAT, they have mechanisms for dealing with the Competing Conventions Problem, but not for solving it. For instance, NEAT has the ability to track genetic innovations, which lets it distinguish different genes and avoid exploring many symmetric cases. It does not completely impede that actual symmetries occur, nor it removes the exploration of different neural networks with same functionality, but it represents a considerable improvement over previous attempts.

Due to their relevance on the field, as well as to their demonstrated ability to deal with game playing problems (Clune et al., 2011, D’Ambrosio et al., 2010, D’Ambrosio and Stanley, 2008, D’Ambrosio et al., 2014, Gallego-Durán et al., 2013, Gauci and Stanley, 2007, Hastings et al., 2009, Hausknecht et al., 2013, Kowaliw et al., 2014, Lehman and Stanley, 2008, Lowell et al., 2011a, Risi et al., 2010, 2009, Stanley et al., 2005b, Stanley and Miikkulainen, 2002a, Suchorzewski, 2011, Tan et al., 2009), NEAT and HyperNEAT are selected for this work. In the case of HyperNEAT, concretely, it even has shown potential for learning to play Atari games directly from visual input (Hausknecht et al., 2012). So, let us detail more closely how they work and which are their main strengths.
and weaknesses.

### 2.2.2 Neuroevolution of Augmenting Topologies

NEAT (Stanley, 2004, Stanley et al., 2005a,b, Stanley and Miikkulainen, 2002a,b,c,d, 2004a,b) is a Neuroevolution algorithm that evolves populations of Artificial Neural Networks starting from the most simple possible topologies and, in Stanley (2004) words, increasingly *complexifying* them. This way of traversing the search space aims at obtaining the simplest possible Artificial Neural Networks that solve a given task. Therefore, NEAT prefers topologies with the least possible hidden neurons. This also contributes to have a minimal number of links and weights. This makes NEAT more powerful, as topologies found have greater change of generalization than other searching methods, because their final associated Vapnik-Chervonenkis dimension (Blumer et al., 1989, Vapnik, 1998) tends to be lower.

Let us explain this last point with more detail. With respect to learning requirements and generalization theory, each link added to a NEAT genome increases its Vapnik-Chervonenkis dimension (Vapnik, 1998)\(^1\). This exponentially increases learning time and exploration required to find a statistically valid solution, according to the Probably-Approximately-Correct (PAC) learning model (Haussler, 1995). This is the main reason why NEAT tries to keep the number of neurons as low as

---

\(^1\)This increase is due to the fact that each link has two weights: one for the link and other for the activation function. These two weights will increase the number of global free parameters of the model, unless they are redundant. And this increase is what ultimately determines a greater Vapnik-Chervonenkis dimension.
possible: the less the number of neurons, the less the number of required links and its associated Vapnik-Chervonenkis dimension.

NEAT is based on a Genetic Algorithm that controls the whole training process. The Genetic Algorithm evaluates populations of Artificial Neural Networks on their fitness at performing the task to be learnt. After the evaluation step, a new population is generated by recombining and mutating the previous population. For this step, NEAT uses a powerful crossover operator and 4 mutator operators:

- **Add a new neuron**: this operator selects one link from a NEAT genome and splits the link in two, adding a new neuron between both links. Then, one of the weights is set to the weight of the previous link, and the other weight is set to 1. This preserves the functionality of the previous link.

- **Add a new link**: this operator selects two neurons from the genome that are not connected and adds a new link between them. The new added link is assigned a weight of 0 to preserve previous functionality.

- **Mutate a weight**: this operator selects a weight from the genome and alters its value by multiplying it by a random factor in the range $[0.5, 1.5]$.

- **Substitute a weight**: this operator selects a weight from the genome and changes its value to a new random value in the range of initial values for weights. This range is a user configurable parameter.
An initial NEAT genotype would contain only the input and output neurons, and some links between them. Starting from that minimal genome, NEAT adds new neurons and links by means of its mutation operators. At the end of each epoch of the Genetic Algorithm, Artificial Neural Network are selected, crossed over and mutated to produce a new population. Leaving aside the idea of starting with a minimal genome, NEAT seems an standard Genetic Algorithm that evolves Artificial Neural Networks. What makes NEAT a special algorithm? NEAT has 3 main innovations that has contributed to Neuroevolution:

- NEAT uses its own crossover operator, specially designed to avoid the Competing Conventions Problem. As stated before, it really does not avoid the problem completely, but its contribution represents a gain of an order of magnitude. Its crossover operator is based on a database that tracks genetic innovation across all individuals and epochs of the Genetic Algorithm. This lets NEAT keep track of the origin of each gene and use this information when mating two Artificial Neural Networks.

- NEAT uses species to protect genetic innovation. Whenever new mutations appear, mutated Artificial Neural Networks tend to perform worse than their parents. Therefore, these mutated individuals have less opportunities to survive and transmit their genetic changes to next generations. However, some innovations will usually require several chained changes to achieve an improvement over previous best performers. Then, protecting innovation through letting new individuals evolve before being discarded increases the
chances of evolution. Speciation is a way to confine genetic modifications in evolutionary niches that protect them from being discarded on early stages of their development.

• NEAT searches through the space of possible topologies using complexification, as it has already been described. This type of traversal maximizes the generalization potential of the solutions found, by minimizing their Vapnik-Chervonenkis dimension.

Figure 2.1 shows an example of a basic NEAT genome. Genome is constituted of neuron genes and link genes. However, neuron genes can be inducted from link genes, because they have information about the neurons they are connecting. That explains why link genes have an InnovationID and neuron genes have not. That InnovationID is a unique identifier for a concrete link. For instance, the link connecting from neuron 2 to neuron 4 is unique for all genomes and for all epochs. Every time a new individual is produced, if it has a link from neuron 2 to neuron 4, that link will be marked with InnovationID 4.

The simple procedure of marking all links produced during evolution with their own unique InnovationID is the basis for the advantage NEAT has over other algorithms. This genetic innovations registry lets NEAT authors develop a new kind of crossover operator. This new crossover operator works by matching genes through their InnovationID. Figure 2.2 shows how this operator works in detail. Once genes from both parents are matched, the crossover operator works as follows:

• If a given gene is present in both parents, descendants inherit the
Figure 2.1: An example of a NEAT genome. Link genes are historically tracked using a unique InnovationID among epochs. Also, when new neurons are added, previous links do not disappear: they become disabled. That lets the crossover operator restore its previous status depending on the other mate being crossed over.

gene from a random parent. It is important to take into account that the weight associated to the link gene will usually be different from one parent to the other. Innovations tracking only takes into account existing links and their associated neurons, but not weight values. Therefore, inheriting from one parent or the other has a significance for evolution. This same behaviour is applied to the enabled/disabled property.

- If one gene is present only in the parent having higher fitness value, the gene is directly inherited by descendants.
- If one gene is present only in the parent having lower fitness value, the gene is not inherited by descendants.
Figure 2.2: Example of a crossover between two NEAT genomes. Right genome has a higher fitness value than left genome. Genes marked in red are those inherited by the descendent.
2.2.3 Hypercube-based NeuroEvolution of Augmenting Topologies

Although NEAT represented a considerable breakthrough, it suffers from the same problem that all direct-encoding\textsuperscript{2} algorithms: they are inherently non-scalable and non-modular. Any algorithm using direct-encoding, including NEAT, has no mechanism for replicating structures or patterns of structures across the phenotype. To explain this with an example, let us imagine an Artificial Neural Network being trained to recognize geometrical forms on images. After some training, a group of neurons and links may be specialized in recognizing an horizontal line. This functionality may be required by several parts of the network as a preprocessing step to recognize higher-level forms. NEAT cannot reuse the group of neurons as a pattern: evolution has to rediscover the pattern as many times as required. This is a great scalability issue, caused by the low-level nature of the mapping between genes and actual neurons/links.

Moreover, closely observing the human brain, there are lots of regularities and patterns that repeat everywhere (D’Ambrosio and Stanley, 2008, Kowaliw et al., 2014, Miikkulainen, 2010, 2013, Stanley et al., 2009). If a direct-encoding scheme was to discover that topology, it should repeatedly discover each one of the regularities again and again. The consequence is clear: direct-encoding schemes are greatly limited to produce large-scale Artificial Neural Network aimed to mimic structures similar to those in human brains. To achieve this goal, a higher-level representation

\textsuperscript{2}A direct-encoding algorithm encodes genomes with literal descriptions link by link, neuron by neuron, weight by weight.
will have better chances of succeeding.

To overcome the impossibility of modularization and pattern replication that NEAT has due to its direct-encoding scheme, Gauci and Stanley (2007) and Stanley (2007) created an indirect-encoding scheme called Compositional Pattern Producing Networks (CPPNs). CPPNs are a kind of networks similar to Artificial Neural Networks, but with an important difference: each node, instead of being a neuron, is a mathematical function\(^3\) (see figure 2.3). Therefore, a CPPN is a composition of functions that can produce outputs full of symmetries, patterns and regularities.

\[ f(x) \]

By describing this composition of functions as a network instead of a formal mathematical composition, the model can profit from existing Neuroevolution algorithms to train and evolve CPPNs. This led to the creation of CPPN-NEAT (D’Ambrosio et al., 2010, D’Ambrosio and Stanley, 2008, Gauci and Stanley, 2007, Stanley et al., 2009): an algorithm that uses a modification of NEAT to evolve increasingly complex CPPNs able to produce different spatial patterns with symmetries and

\[^3\text{Typically, a continuous function like Sine, Gaussian or Absolute Value, but it could be any desired function.}\]
regularities.

HyperNEAT (Gauci and Stanley, 2010, Risi et al., 2010, Stanley et al., 2009) takes NEAT and CPPNs as indirect-encoding scheme and produces large scale Artificial Neural Networks with regularities, patterns and symmetries. HyperNEAT takes a population of CPPNs\(^4\), and uses CPPN-NEAT to evolve them.

For a CPPN to produce an ANN, a geometric *substrate* is required. A substrate is a collection of nodes (i.e. neurons) placed in a \(N\)-dimensional space, that have a vector of coordinates \(\mathbf{x}^i = (x_1^i, x_2^i, ..., x_n^i)\) for each node \(i\). Typically, in a 2D-space neurons would be scattered in \([-1, 1] \times [-1, 1]\).

Once a substrate is defined, the next step is to add links and weights between neurons. This is done iteratively querying the CPPN with the coordinates of each possible pair of neurons \((\mathbf{x}^i, \mathbf{x}^j)\)\(\forall i, j\), where the output value from the CPPN represents the weight of the link from \(\mathbf{x}^i\) to \(\mathbf{x}^j\). With these queries, CPPNs produce spatial patterns that HyperNEAT interprets as connectivity patterns among neurons from the substrate (Gauci and Stanley, 2010). This is the ways by which CPPNs produce large-scale Artificial Neural Networks. As CPPNs are generally composed of continuous functions, connectivity patterns can be used in different granularities. This lets HyperNEAT produce Artificial Neural Networks of varying number of nodes, but with the same regularities and connectivity patterns.

\(^4\)Considering CPPNs as genotypes. Then, they become an indirect-encoding scheme, as they are used later on to produce their associated phenotypes: actual large-scale Artificial Neural Networks. These phenotypes are the ones used finally to solve the task HyperNEAT is being trained for.
Taking into account that links and their weights are produced as a function of the relative location of neurons in space, it follows that the resulting topology of the Artificial Neural Networks has to be related to the actual geometry of the substrate. This is an interesting characteristic of HyperNEAT, because it can produce Artificial Neural Networks with the ability of understanding geometry relations among their inputs. This feature, combined with the ability of HyperNEAT to produce same patterns with manageable degrees of granularity, have incredible interesting applications. Both features may be exploited by problems in which geometry relations among input have great significance. For instance, thinking about a chess controller with 64 inputs (one for each square of the board), CPPNs can produce Artificial Neural Networks with intrinsic knowledge of the board structure encoded in the connectivity pattern. This feature is not present in traditional Artificial Neural Networks, which have to discover this information by themselves during training.

Moreover, HyperNEAT also has the ability of having a relatively lower Vapnik-Chervonenkis dimension (Blumer et al., 1989, Vapnik, 1998) whereas dealing with large-scale Artificial Neural Networks. Similarly to what happens on the training process of a Support Vector Machine (Cortes and Vapnik, 1995), HyperNEAT does not explore the full space of possible Artificial Neural Networks. HyperNEAT only explores parts that can be encoded using selected functional patterns: repetitions, symmetries, sequences... These functional patterns are determined when selecting mathematical functions for the CPPNs. Ultimately, CPPNs are networks whose nodes represent mathematical functions. Therefore, as
2.3 Measuring and predicting difficulty

In training and education, difficulty plays a key role in the learning process. In order for optimize learning, difficulty of any given exercise should match abilities of the learner (Koster and Wright, 2004, Ley and Kump, 2013, Petkovic et al., 2012, Schalk et al., 2011, Yoo and Kim, 2014). Matching the abilities means being so difficult as to be an interesting challenge, at the same time as being so easy to be reachable with a limited amount of effort in time. Learners presented with exercises that do not match their abilities have greater probability of abandoning.

Therefore, correctly estimating the difficulty of learning activities is the first step to be able to optimize the learning process. Some research work has been carried out calibrating difficulty by analysing student historical data (Ravi and S., 2013), or using linear regression to estimate difficulty based on user data (Cheng et al., 2008) or even on generating exercises automatically with a given established difficulty (Radošević et al., 2010, Sadigh et al., 2012). But these studies are spread, discontinued and seem to be disconnected from each other. In general, the concept of difficulty within the academic world does not seem to capture
too much attention.

More studies related to difficulty can be found changing the focus to the field of Computer Games. Researchers in this field seem much more concerned about the importance of difficulty. The parallelism with academic learning is complete: if a level of a game is too difficult or too easy, players tend to stop playing the game. Therefore, it is vital for a game to have a well designed progression of difficulty, if willing to catch the attention of the players. Most studies in this field try to develop methods to dynamically adjust difficulty to match player’s skills (Hunicke, 2005, Hunicke and Chapman, 2004, Missura and Gartner, 2011, Mladenov and Missura, 2010). All these studies use existent levels of difficulty proposed in present Computer Games and focus on selecting the most appropriate for each player and game being played. Hunicke and Chapman (2004) and Hunicke (2005) take measures of performance of the player and tries to predict if the player is going to fail to anticipate and adjust the level of difficulty. The proposal is completely specific to First Person Shooter (FPS) games (Saldana et al., 2012), as measures are defined for this specific type of gameplay. Mladenov and Missura (2010) use data collected from previously played games to analyse a set of gameplay characteristics and input this data to a supervised Machine Learning algorithm. The goal is to have an offline prediction of the level of difficulty players are going to select in their next game. Missura and Gartner (2011) take a different approach for automatically selecting difficulty for a given player among a set of finite difficulty levels. They divide the game into play-review cycles. They measure the performance
of the player in the play cycles, and change difficulty level on review
cycles accordingly to their estimations.

Herbrich et al. (2007) present a very interesting work on measuring
players’ skills comparatively. Their system, called TrueSkill, is based
on chess’ Elo rating system (Elo, 1978). Just like the Elo rating system,
players have a 1-dimensional value ranking that predicts their probability
of winning against other players by logistic comparison. Although this
work is not directly based on difficulty, it is indirectly valuing players’
skill with similar intention: match players against those with similar
abilities to foster balanced games. It is considerably interesting because
this system has been applied during several years to XBOX 360 live
players for different kinds of games.

Another interesting work is that proposed by Mourato and dos Santos
(2010). Their goal is to procedurally generate content for Platform Games
similar to Super Mario Bros (Pedersen et al., 2009). The problem with
this kind of content is how to classify the generated content with respect
to difficulty. They propose a way to measure difficulty in Platform Games
by measuring players’ probability of failing after each individual obstacle
in the game. Presented concepts are interesting but they lack a practical
result with actual players and ready-to-be-played generated content.

Finally, Aponte et al. (2009) present one of the most interesting re-
viewed works. In their work they state that their goal is “to evaluate a
parameter or a set of parameters that can be considered as a measure of
a game difficulty.” They start by measuring the difficulty of a reduced
Pacman game with 1 ghost. In their Pacman game, speed of the ghost is
a configurable parameter to make the game more difficult at will. They measure the score of a synthetic player as number of eaten pellets and then show a graph with the evolution of this value depending on the speed of the ghost. This approach lets them show the progression of difficulty depending on the selected level (speed of the ghost). Based on that result, they define a set of properties that a general definition of difficulty should have, and propose a general theoretic definition of difficulty as the probability of losing at a given time t. They only propose this definition, but do not perform any kind of test or mathematical proof. It ends up as a simple proposition based on their arguments.

After analysing the state of the art, it has been impossible to find any work that ever tried to predict difficulty of any kind of activity at design stages. Existing works either measure difficult “a posteriori”, based on collected data, or try to modify it to match present or past status of a player. Moreover, only one work was found that deepens a little more into defining difficulty (Aponte et al., 2009). However, this work presented very basic experimental results not related to the definition of difficulty, and did no test at all of their proposed definition. Moreover, their proposed definition is so theoretic that seems to be not possible to apply to a real world scenario.

Some of the previous works analysed used Machine Learning for predictions, but neither of them questioned whether a potential correlation exists between Machine Learning and humans, with respect to the way of learning. Only the work of Griffiths (2009) has some insights on this relation, but from a completely different point of view. In their work,
Griffiths (2009) states that Probabilistic models of cognition provide a set of tools for characterizing human learning. They expect these models to provide a good basis for Machine Learning practitioners to develop new systems based on human cognitive biases. However, they do not explore correlations in learning progression between humans and nowadays Machine Learning algorithms.

This work will focus on all these fields that have been left unexplored by previous works, and will explore the relation between Neurovolution (using NEAT and HyperNEAT) and humans with respect to their way of learning.
Chapter 3

Measuring Difficulty of Learning Activities

Difficulty is a widespread concept used to intuitively measure required abilities and cost for a learner to successfully complete a given learning activity. In spite of being subjective, it is considered among the main factors determining learners’ motivation (D’Mello et al., 2012, Domínguez et al., 2013, Lee and Hammer, 2011, Verdú et al., 2012, Wang and Newlin, 2000). This is best exemplified with the notion of The Flow Channel (Getzels and Csíkszentmihályi, 1976, Schell, 2008) (see figure 3.1). The Flow Channel represents the way difficulty and skills of the learner relate to each other:

- When difficulty is much higher than learners’ skills, anxiety appears. This is psychologically explained by learners perceiving their
skills as insufficient, thus getting demotivated. They normally feel that the activity requires too much effort compared to what they think they could do. This often leads to early abandon.

- On the contrary, if learners’ skills already include what the activity provides as learning outcome, boredom shows up. Having to invest time and / or resources to get an already possessed outcome is interpreted as lost time. Interest vanishes, motivation decreases and boredom appears.

- When skills and difficulty are balanced, learners enter a state of Flow. In Schell (2008) words, “Flow is sometimes defined as a feeling of complete and energized focus in an activity, with a high level of enjoyment and fulfilment.”

Figure 3.1: Representation of the flow channel, by Csikszentmihalyi (1990)

This research assumes The Flow Channel theory as a key point for
improving the design and selection of learning activities. So, it follows that objective and useful definitions for difficulty and skills are important. Literature shows some attempts at creating definitions for difficulty (Aponte et al., 2009, Missura and Gartner, 2011, Mourato and dos Santos, 2010), showing most recent interest coming from research in Computer Games. However, these definitions do not have the degree of generality and meaning required for the aims of this research. They are either too specific (Mourato and dos Santos, 2010, Ravi and S., 2013), have a non-general meaning (Missura and Gartner, 2011, Mladenov and Missura, 2010, Mourato and dos Santos, 2010), are not evaluable for different sets of inputs (individual users, groups, etc.) (Hunicke and Chapman, 2004), or do not take time and learners’ evolution into account (Mladenov and Missura, 2010, Mourato and dos Santos, 2010).

Let us start by identifying the information sources available for measuring difficulty, together with their features and limitations. Then derive desired properties for a useful and objective definition of difficulty. Next, continue designing some mathematical functions that meet the properties and give useful outcomes. Finally, test designed functions and analyse their usefulness for learning activities.

### 3.1 Sources for measuring difficulty

Let us consider difficulty as a cost: in order to successfully finish an activity, any learner has to pay a cost in time and effort. Measuring time is trivial from a conceptual point of view. The problem comes from
measuring effort. How can we measure effort? Do we have an objective
definition of what effort is?

Let us consider an example of activity: “scoring five 3-point shots in
a basketball court, in less than 5 minutes”. This is a training activity
whose expected learning outcome is an improvement in shooting precision
to basket\(^1\). This activity will take at most 5 minutes, and at least the
time required to shot 5 times: time cost is straightforward. Regarding to
effort, it will depend on previous conditions. A trained, muscular player
may complete the activity fast, without much effort, whereas a weak
novice could require many attempts to finish it successfully. Moreover,
novice players may waste much more energy because they lack adequate
technique. This could also be considered more effort.

The activity could be analysed many times and from different perspec-
tives, and many definitions for “effort” could be found. Before entering
an endless debate on what “effort” is or should be, let us consider a useful
point of view with respect to our goal of measuring difficulty. An indirect
measure for “effort” could be derived from the intrinsic failure / success
measures of the activity. When 5 minutes are over, a player that scored
4 baskets is closer to success than other who only scored 1. It can be
considered that having scored 4 baskets leaves out less progress to be
done for succeeding than scoring just 1. Under this consideration, there
is less effort pending to succeed when more percentage of the activity has
been completed.

\(^1\)Although other learning outcomes can be considered from this activity, let us
consider it just as a precision improvement exercise
The previous argument considers that effort is indirectly related to progress. The more progress is achieved, less effort is required to finish. Although this logic consideration is not a concrete definition of effort, this point of view has many advantages:

- For many kinds of activity, progress is relatively easy to define and measure objectively.

- A measure for progress is also closely related to learning outcomes: most activities yield learning outcomes even when not fully completed. In fact, that learning outcomes become clear when success ratio increases out of repeating the activity.

- As progress to success is one of the key factors in motivation, measures taking progress into account also foster motivation.

Therefore, this research will consider “more difficult” an activity when less progress is done. In the sake of rigour, progress will be considered with respect to time: progress percentage per unit of time will be an inverse measure for difficulty. So, an activity being “more difficult” will imply that less progress is made per time unit. This will let us measure difficulty in an intuitive, understandable and objectively measurable way.

### 3.2 Desired properties for difficulty

There are potentially infinite ways of defining difficulty as a relationship between time and progress. It is important to have guidance for selecting an appropriate measure from such a huge set of potential definitions. So,
establishing a set of desired properties will ensure that the selected definition is useful under defined criteria. Moreover, these desired properties will act as restrictions, reducing the search space.

Let us consider the next set of properties, having present that measuring and comparing learning activities is the final goal:

- Difficulty should always be positive. Progress and time are always positive or 0 values when measuring a learning activity. A negative difficulty coming out of these two values is impossible and would have no meaning.

- Difficulty should have a minimum value. A difficulty value of 0 would mean that no time / effort is required to finish a given activity. That would correspond to an activity that is already done.

- Difficulty should also have a maximum value. Making difficulty unbound would imply that any value could be “not so difficult” compared to infinite. Having a maximum value lets us fix impossible activities, which is desirable. Let us think about scoring 5 3-point baskets on 0.01 seconds. That is not extremely difficult, but impossible\(^2\). With an unbound upper limit that should be labelled as infinity, which makes formulation more complicate and has no advantage on comparisons (Would a value of 1000 be very difficult or insignificant?).

- Fixing 1 as the maximum value for difficulty has advantageous

properties. That bounds difficulty in the range \([0, 1]\), which lets us consider it as a probability. That makes sense and is compatible with previous considerations. Moreover, that enables the probability theory as a valid set of tools for working with difficulty, which is very desirable.

- Difficulty should not be a unique value but a function over time. While an activity is being done, difficulty keeps changing as progress is being made.

- Difficulty must be a continuous function over time. It makes no sense for a moment in time not to have a difficulty associated.

- Difficulty must be a non-strictly decreasing function. Every time a learner makes progress on a given learning activity, difficulty decreases by definition as less progress is required to meet success.

Let us compose a function with these properties, following the basketball example. Let us imagine a player that scores 5 baskets at times \(t_i \in \{15, 40, 62, 128, 175\}, \ i \in \{1, 2, 3, 4, 5\}\) in seconds. Difficulty could be represented as shown in figure 3.2: whenever player scores baskets, difficulty decreases. Decreasing difficulty can be considered as a step function, maintaining its value except on scoring events. It also can be considered as a linear function, which results on a much smooth shape. Moreover, a linear function seems to inform better about the pace of the player.

As it can be deduced from figure 3.2, these properties configure a very powerful definition of difficulty: it goes far beyond a simple scalar quan-
Chapter 3. Measuring Difficulty of Learning Activities

Figure 3.2: Manually constructed difficulty function for basket example. Difficulty decreases as player progresses, scoring baskets in this example.

Let us define the previous basketball example a little bit differently. This time, the activity will be “scoring the maximum possible number of 3-point shots in a basketball court, in 5 minutes”. At a first glance it looks pretty similar to the previous definition, and even a little bit more challenging. However, this activity poses a problem when using desired
3.3. **Intrinsic limitations**

properties to measure difficulty over time: there is no theoretical maximum value for score \(^3\). Moreover, there is no measure for success: there is no way to state that a player has been successful in the exercise or not. Although the exercise lets us compare results from different players, there is no way to measure progress towards completion, as there is no completion status defined.

The properties selected limit the way activities should be defined. Not every possible activity will fit for this model. This is both a limitation and a design guide. Activities designed for this model of difficulty will have a set of properties:

- Activities require progress to be measurable (i.e. they should have a score). For instance, an activity defined as “selecting the proper answer from a set of 4” has no way of measuring progress. Although time to answer and success can be measured, there is no progress towards success. Resulting functions would represent either a full square or a line, depending on model selected. That would not give the same information as in the basketball case.

- Score (i.e. progress) has to be non-strictly increasing function over time. As score is measuring progress to an end it does not make sense for it to decrease. For instance, any new basket scored in the basketball example makes a positive score, and there is no way to revert it or add a negative score. General score measures having

\(^3\)In this case, a practical maximum could be set considering human limitations in the total number of possible shots during 5 minutes. However, this may not be applicable to other examples.
punishments or negative score events would not be appropriate for this model. However, almost any general score measure could be transformed in an equivalent non-strictly increasing measure for this purpose.

- Activities must have a measurable success status or, at least, a maximum score. This status is required to define difficulty within its limits. Progress can be measured in unbounded activities, but cannot be scaled to a $[0, 1]$ range.

- Activities must be considered over time. For instance, an activity about creating a program cannot be considered just as its final result. Having a single point of evaluation is similar to not being able to measure progress. Moreover, it is very important to measure time required to do the activity. If all the learners hand the result of an activity at the same point in time and no measures have been taken previously, no data will be available for the model.

These intrinsic limitations are part of the selected set of properties and shall be assumed. However, it can be seen with a positive perspective. Having activities where progress is measurable over time and with well defined score limits or success status is very interesting for learners. Progress informs learners about the status of their evolution to success, and also remove their doubts about their skills being enough for the activity. Although these design impositions are not easy to achieve on every activity, they are definitely desirable from educational point of view.
3.4 Mathematically defining difficulty

With all desired properties and limitations clarified, a working mathematical definition of difficulty can be constructed. Let $A$ be the set of all possible activities, and $L$ the set of all possible learners. Let $\alpha \in A$ be a concrete learning activity. As an activity, $\alpha$ can be realized by any learner $l \in L$. Each $l$ realizes $\alpha$ a number of times $N_l \in \mathbb{N}$. So let $\alpha^i_l, l \in L, i \in \mathbb{N}, i \leq N_l$ represent the $i$-th realization of the activity $\alpha$ by the learner $l$.

Each $\alpha^i_l$ takes an amount of time $t^i_l \in \mathbb{R}$, measured in seconds. Let us consider, for simplicity, that each $\alpha^i_l$ starts at time 0 and ends at $t^i_l$. Then, let $S_t(\alpha^i_l) \in \mathbb{R}$ be a function that measures the score got by learner $l$, at time $t$ on its $i$-th realization of $\alpha$. So, $S_t(\alpha^i_l)$ is the function that measures the progress towards success of a learner that performs an activity.

The score function is expected to be explicitly defined for each activity. In fact, a potentially infinite number of score functions can be defined for each activity. Therefore, let us assume that activities and their score functions are defined by activity designers. Also, for clarity reasons, let us assume that activities and score functions meet desired properties and limitations exposed on sections 3.2, 3.3.

In previous sections, difficulty has been defined as the inverse of progress. However, this cannot be defined exactly this way. Difficulty must be defined in $[0, 1]$ range, and the score function could have a much broader range. However, the score function should be non-strictly increasing, and should have an upper limit. Therefore, the score function
could be safely assumed to start at 0, because the actual range of the function can always be moved to start at 0. Let $S^*(\alpha)$ be the maximum score value for the activity $\alpha$,

$$S^*(\alpha) \in \mathbb{R}, \quad S^*(\alpha) \geq S_t(\alpha_i) \quad \forall l \in L, i \in N_l \quad (3.1)$$

This lets us define the “easiness function” as a scaled version of the score function over time in the $[0, 1]$ range:

$$E_t(\alpha_i) = \frac{S_t(\alpha_i)}{S^*(\alpha)} \quad (3.2)$$

The function defined in equation 3.2 is called “easiness function” as it is exactly the inverse of the initial definition of difficulty. Therefore, the definition of difficulty follows:

$$D_t(\alpha_i) = \frac{1}{E_t(\alpha_i)} \quad (3.3)$$

### 3.5 Improving the definition of difficulty

This definition of difficulty is tied to the concept of progress. It represents an advantage over estimating difficulty with just a single scalar value: the resulting graph shows an evolution over time which informs of the whole realization of the activity. It also yields instant values for difficulty at any time of the realization. This values intrinsically represent the percentage of progress remaining to finish the activity. They could also be interpreted
3.5. *Improving the definition of difficulty*

as the probability of failing the activity.

However, these values are quite plain: they are instant values that do not capture information on the progress by themselves. The result is similar to considering any instant $t$ to be independent from the others that compose the timeframe of the activity. For instance, this is like considering in the basketball example that scoring at first shot is equally probable to scoring after 4 baskets, or at a last attempt, when time is finishing. Nevertheless, a more accurate definition should consider that events occurring at time $t$ are influenced by all events happened in the range $[0, t]$.

Experience shows that influence of a timeframe over next time steps is strong on humans. It is convenient to consider how human factors relate over time: psychological status, strength, fatigue, motivation, etc. Time steps in the timeframe of any learning activity, performed by a human learner, are best considered to be strongly interdependent. Therefore, can be improved by making $D_t$ depend on a function of all $t' \in [0, t]$, to make final values express this interdependency.

There are infinite potential approaches to make $D_t$ dependent on its “history". Moreover, there is no theoretical way to determine the appropriate way to weight all the possible factors. What is more, different activities and learners will have different influence factors. This makes extremely difficult, if at all possible, to design a theoretical relation covering such a chaotic landscape. This suggests using an experimental

---

4 This interpretation is bound to discussion about its real meaning as a probability.

5 Note that the term “history” is a convenient way of referring to the set of all past values of difficulty $\{D_{t'} / t' \in [0, t]\}$
approach instead. Therefore, this research starts modelling influence in a very simple way. This first model can be used as a benchmark to test other different approaches and experimentally determine better ways of defining difficulty.

Assuming that \( D_t, \forall t \) should depend on \( \{D_{t'}/t' \in [0,t]\} \) and \( 0 \leq D_t \leq 1 \), let us define \( D_t \) as the area of the curve above \( E_t \) related to the maximum possible area up to the instant \( t \),

\[
D_t(\alpha_i^l) = 1 - \frac{1}{t} \int_0^t E_t(\alpha_i^l) dt
\]  

(3.4)

Equation 3.4 defines difficulty \( D_t \) as a value depending on all previous history of the \( i \)-th realization of an activity \( \alpha \) by a learner \( l \). The dependency is made indirect, using the easiness function as a proxy for difficulty. This makes definition easier, eliminating recursive references and associated problems.

Using the new definition stated at equation 3.4 the graphical layout of \( D_t \) varies greatly, as figure 3.3 shows. Compared to figure 3.2, the new definition for \( D_t \) results in a function that responds much smoothly to score events. This new behaviour shows an interesting feature. Let us assume that \( t \in [0,t^*] \). Using equation 3.4, \( D_t^* \) will directly depend on the performance shown by the learner during the realization of the activity (being \( D_t^* > 0^6 \)). In the basketball example, the faster baskets get scored, the lower \( D_t^* \) will be, and vice-versa. Therefore, after completing an activity, the lower the residual difficulty value \( D_t^* \), the greater the

---

6Unless \( D_0 = 0 \), which would only happen on activities completed at start time. That is a degenerate case with no interest in practice. Thus, it can be safely ignored.
3.5. Improving the definition of difficulty

performance shown by the learner.

Figure 3.3: Behaviour of $D_t$ using equation 3.4 with data from the basketball example from section 3.2. Left, exact definition for $E_t$ with step value changes. Right, linear interpolation for $E_t$.

The interesting property shown by $D_{t^*}$ is a direct consequence of its cumulative definition. So, this property will be shown by $D_{t'}, \forall t' \in [0, t^*]$. Therefore, $D_t$ can now be used as a performance measure with more information than $E_t$, as it integrates information about score and time / frequency in one single value. Careful analysis of $D_t$ for different learners and realizations of the same activity could lead to establishing correlations with abilities learnt and degree of mastery. Next chapters will show $E_t$ and $D_t$ graphs elaborated with real data and analyse this property further.
Chapter 4

PLMan: A Game-based Learning Activity

Section 3.3 states the intrinsic limitations resulting from the proposed definition of difficulty. The potential advantages of this definition require activities to be designed in a concrete way imposed by these intrinsic limitations. In order to experiment with the definition of difficulty, this chapter introduces an activity designed with the needed requirements: the PLMan game.

PLMan (Castel De Haro et al., 2009, Gallego et al., 2014, Villagrán-Arnedo et al., 2009) is a game originally designed to be used as learning activity at the University of Alicante. The game challenges students to solve some Pacman-like mazes by means of logic programming in Prolog (Wielemaker et al., 2012). Students populate Prolog knowledge bases with facts and rules designed to programmatically control Mr.PLMan,
a Pacman-like character. Their goal is to create Prolog programs that automatically guide Mr.PLMan through mazes, eating all the dots to succeed.

Next sections describe the most important aspects required to understand what is PLMan and how it works. There are many other aspects regarding PLMan and its use as learning tool that are not crucial to the main claim of these work, but may be of interest to the reader. All these aspects are thoroughly described in appendix A.

4.1 PLMan: The game

In PLMan, as stated before, students create automated controllers for Mr.PLMan. The goal is making these automated controllers able to eat all the dots of a given maze, dodging the perils. Automated controllers are developed in Prolog programming language, constructing sets of rules to reason about the maze and decide actions. Each time students develop any new controller for a given maze, they are automatically assessed. If their controller is successful, they are automatically presented with the next maze to continue. An example maze along with an automated controller written in Prolog is shown in figure 4.1.

Mazes are designed to have an increasing difficulty, requiring progressively more programming abilities. In the first mazes, simple rules in the form “If you see an enemy to your left, move right”\(^1\) are enough to construct successful controllers. As the game progresses, more difficult

\(^1\)in Prolog: `plman :- see(normal, left, 'E'), doAction(move(right)).`
4.1. PLMan: The game

Mazes are delivered, requiring more complex controllers to succeed. This leads students to learn Prolog programming, as well as logic thinking and small bits of Artificial Intelligence.

Figure 4.1: Example maze along with the Prolog knowledge base that controls Mr.PLMan (@) to eat all the dots dodging the enemy (E)

PLMan is a turn-based game. It works similar to a classic board game, but in an electronic way. At the start of each turn, the game transfers the control to the controller (the Prolog program created by the student). Then, the game waits indefinitely\(^2\) until the controller selects the next action to be performed. When the controller returns the selected action, the game updates one complete turn (i.e. moves or changes the status of other entities in the maze) and starts the next turn. Figure 4.2 shows this cycle.

Mr.PLMan can only perform one single action at each turn, and it is able to carry one object at a time. The set of possible actions that Mr.PLMan can perform at each turn is limited to 4 generic actions:

- **move(Direction)**: Mr.PLMan moves 1 cell in the specified orthogonal direction (up, down, left, right). Optionally, **move(none)** may be used to stay in place for 1 turn.

---

\(^2\)On real executions, maximum amounts of time are defined to prevent infinite loops and to encourage minimally efficient code.
• **get(Direction)**: Mr.PLMan gets one object placed at one of the 4 contiguous orthogonal cells. **get(here)** can also be used to get one object placed in the same cell PLMan is occupying.

• **drop(Direction)**: Reverse action for **get(Direction)**, dropping the object Mr.PLMan is carrying.

• **use(Direction)**: Mr.PLMan uses the item it is carrying towards one of the 4 orthogonal directions.

The game ends when some of these criteria is met (Success status is shown in brackets):

• (Success) All the dots in the maze are eaten by Mr.PLMan.

• (Fail) Mr.PLMan dies due to interaction with a mortal entity (an enemy, bomb, trap, etc.).

• (Fail) The limit of turns for the given maze is reached.

• (Fail) There is a time-out during execution.
The score the automatic controller gets is the percentage of dots eaten. Additionally, some punishments are also added to the score to enforce testing, code revision and detailed behaviour design. Punishments are applied to score each time one of this events happen (name of the event in brackets):

- **(Collision)** Mr.PLMan tries to move to a cell occupied by a solid entity.

- **(Invalid action attempt)** Mr.PLMan tries to get an object from a place where no object is placed, or tries to drop / use an object when having none.

- **(Erroneous action)** Mr.PLMan tries to use an object in an erroneous way. For instance, trying to use a key object into a wall instead of a door results in an erroneous action.

- **(Rule failure)** Mr.PLMan’s controller fails to select an action to perform. None of the clauses of the control rule defined in the knowledge base is successful in a given turn.

The final score is calculated as the main score (percentage of eaten dots) minus the punishments (one single quantity for each punishable event that happened). For any given maze, the maximum score is 100%, and the minimum one is 0% (no punishments are applied beyond 0%).
4.2 Developing controllers for mazes

To construct a successful controller that beats a given PLMan’s maze, collecting information about Mr.PLMan’s environment at each turn is essential. As the mission of the controller is to decide the action to be performed, it requires to reason about the environment. As PLMan is a game focused on Artificial Intelligence perspective, the controller only has partial information available. This information is provided to the controller through virtual sensors that are defined for Mr.PLMan.

There are two sensors that collect the input information the controller can access. First is the normal sensor: a short-range visual sensor. This sensor provides the information about the visual appearance of the entities located at most 1-cell away from Mr.PLMan. It gives 9 inputs to the controller at each turn. Figure 4.3 shows exactly how it works. PLMan defines a predicate `see/3` which may be used by the controller to check whether there is a given element at one of the 9 cells surrounding Mr.PLMan.

```
see(normal, left, L) --> L='.'
see(normal, right, R) --> R='.'
see(normal, up, U) --> U='E'
see(normal, down, D) --> D='.'
see(normal, here, H) --> H='.'
see(normal, up-left, UL) --> UL='#'
see(normal, up-right, UR) --> UR='.'
see(normal, down-left, DL) --> DL='.'
see(normal, down-right, DR) --> DR='a'
```

Figure 4.3: PLMan’s short-range visual sensor. Mr.PLMan (@) is able to see two walls (#), an enemy (E), a dot (.), an object (a) and some empty spaces ( ).

---

3 For technical details about defined predicates, see section A.6
4.2. Developing controllers for mazes

The second sensor is a long-range visual sensor called the list sensor. This sensor provides information about visual appearance of entities that are in one of the 4 Mr.PLMan’s orthogonal lines of view. The list sensor is limited by the nature of entities: it does not give information of entities shadowed by any previous solid entity (a wall, a door, etc.). Figure 4.4 shows the list sensor in action: at current Mr.PLMan’s status, the controller can get information about 3 cells to the left, 2 to the right, 3 downside and 5 upside. The information is given to the controller as a list containing the visible entities presented in order of sight. This information is determined by nearest solid objects in Mr.PLMan’s line of sight: there are 4 walls that impede Mr.PLMan’s to look beyond. Therefore, the amount of information given by this sensor will vary greatly depending on Mr.PLMan’s location.

![Figure 4.4: PLMan’s long-range visual sensor](image)

The information provided by these 2 sensors can be used to reason about the environment and decide actions to take. For instance, simple rules in the form “if the object in the next cell to the right is a dot, move Mr.PLMan to the right” are easily constructed using a combination of see/3 and doAction/1 predicates:

```
see(list, left, L).
  --> L = [ '.', 'E', '#' ].

see(list, right, R).
  --> R = [ 'E', '#' ].

see(list, up, U).
  --> U = [ '.', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#' ].

see(list, down, D).
  --> D = [ ' ', ' ', ' ', ' ', ' ', ' ', ' # ].
```
control_rule:-see(normal, right, '.'), doAction(move(right)).

Let’s analyse the rule: when a turn starts, PLMan launches control_rule, which is defined in the knowledge base of the controller. control_rule requires 2 conditions to be met for succeeding: first, a dot must be placed 1-cell to the right of Mr.PLMan. If this first condition is met, the second one will be launched, effectively selecting “move right” as next action. Then, execution of control_rule will end with a success state. On the contrary, if the first condition is not met, the second one will not be launched, and this clause of control_rule will end with a failure status. If more clauses of control_rule are defined, the program will continue examining them. Otherwise, the execution of the controller ends with a failure status, yielding a warning to the user and a punishment in the score.

This is the basis for defining a controller with the desired automated behaviour for Mr.PLMan. Using Prolog programming techniques, any kind of computable way of selecting actions can be developed. Interestingly, mazes can be designed in a way so as to encourage students to use progressively complex programming constructions. Initial mazes can be simple enough as to be solved with 2 or 3 simple clauses of a rule like the one presented on the previous paragraph. Complex mazes may require advanced Prolog programming to create controllers able to have their own internal state (being capable of remembering data using a memory) and analysing different sources of information to develop and follow plans, and not only reacting instantly to given stimuli.
4.3 Examples of developed controllers

This section shows how the PLMan game performs concrete actions selected by controllers. Some maze examples have been selected to clarify these mechanisms. About the examples shown, it is important to know that mazes are classified by teachers in 5 stages. Stages isolate mazes that require some specific knowledge to construct a successful controller. The greater the stage, the more complex the required knowledge to develop controllers. Within each stage, mazes are classified by their difficulty from 1 to 5. Experts measure difficulty on the base of the time they require to develop a 100%-scored controller.

Let us start with the simple example shown at figure 4.5. This example shows a maze with some corridors full of dots that take some turns. Solving this maze requires giving instructions to Mr.PLMan to move in the direction in which next dot is seen. This is easily translated into Prolog as simple clauses of a rule like the ones presented at the bottom of figure 4.5. When these clauses are executed, they make Mr.PLMan move. These movements are represented in 4.5 showing 6 concrete steps of the execution of the game. The brown line shown at some of these 6 steps represents the movements made by Mr.PLMan since its previous step.

Step 1 from figure 4.5 shows the initial state of PLMan in the maze. At that turn, first clause of the rule successfully checks for a dot at the first cell to the right of Mr.PLMan (\texttt{see(normal, right, '.')}). Then execution moves to \texttt{doAction(move(right))} selecting “move to the right” action. So, game changes to step 2. This behaviour repeats 4 more
Chapter 4. PLMan: A Game-based Learning Activity

Figure 4.5: An example of a solved maze from PLMan’s stage 1. The turns, ending at step 3 from the figure. Then, the first clause of the rule fails, as no more dots are found to the right. Execution continues to the second clause, which succeeds as a dot is found 1-cell down Mr.PLMan. Next 2 turns yield the same result, ending at step 4. Then, next 4 turns move Mr.PLMan to the right as clause 1 succeeds again, arriving to the corner. Next, clauses 1 and 2 fail; execution analyses clause 3, which succeeds as dots are 1-cell up Mr.PLMan. This happens 3 more times, ending at step 5. Execution continues with similar analyses until step 6, where all dots are eaten and the game ends with a 100% score.

Figure 4.6 shows a more complex example. In this case, there are several objects scattered around the maze. There are 2 keys (v: green key, r: red key) two doors (l) and a patrolling enemy ‘E’. The enemy moves from left to right and vice versa, staying still for 3 turns each time it collides with a solid object (a door or a wall). To solve the maze, Mr.PLMan has to get one key, open the corresponding door, leave the
key, get the other key and open the other door. This is necessary to eat all the dots, as 4 of them are behind the doors.

When conditions and movements increase, complexity of controller’s code grows too and more knowledge about Prolog programming is required to create good quality solutions. Solution proposed this time has all conditions as data stored in Prolog facts. This data is read through the main rule which only has one clause this time. This clause uses Prolog’s backtracking mechanism to iterate through all the data. Three predicates have been created to simplify condition checking: \( c(D, O) \) checks for object \( O \) being at the cell in the direction \( D \) (an alias for \texttt{see/3}), \( o(A) \) checks for Mr.PLMan having the object \( A \) in its inventory and \( no(A) \) checks for Mr.PLMan not having the object \( A \). Finally, another predicate \( \text{cond}(L, A) \) stands for a check list \( L \) of conditions that must be met in order for action \( A \) to be selected.

### 4.4 PLMan as learning activity

The design of PLMan as an Artificial Intelligence programming game confers some important properties to it as a learning activity. First, solving a PLMan maze is an activity that takes a considerable amount of time. Students have to analyse the maze and mentally design a way to traverse it eating all the dots. After an initial analysis, they start designing and testing basic controllers with some initial rules. Then a testing-redesign cycle in small steps lets students advance step by step towards a successful controller for the maze. Many intermediate controllers are created,
Figure 4.6: An example of a solved maze from PLMan’s stage 2.

```prolog
:- use_module('pl-man-game/main').
cond([] c(down, '.'), move(down)).
cond([no('r')] c(up, 'E'), move(nothing)).
cond([] c(up-left, 'E'), move(up)).
cond([o('v')] c(right, '.'), move(right)).
cond([] c(right, 'r'), use(right)).
cond([] c(down, 'r'), c(right, 'v'), get(down)).
cond([o('v')] c(down, 'r'), drop(right)).
cond([] c(left, '.'), move(left)).
cond([] c(right, '.'), move(right)).
cond([] c(right, 'v'), move(right)).
cond([] c(left, 'v'), get(left)).
cond([] c(up, '.'), move(up)).
cond([o('v')] c(left, '.'), move(left)).
cond([] c(left, 'v'), move(left)).
cond([] c(right, '.'), move(right)).
cond([] c(up, '.'), move(up)).
cond([] c(left, '.'), move(left)).
cond([] c(down, '.'), move(down)).

rule:-
cond(CL, ACT),
forall(member(c(D,O), CL), see(normal, D, O)),
forall(member(o(X), CL), havingObject(appearance(X))),
forall(member(no(X), CL), not(havingObject(appearance(X)))),
doAction(ACT).
```
normally achieving higher scores as they progress in the testing-redesign cycle.

This creation cycle makes students steadily increase their knowledge about the maze and improve their controllers. As knowledge increases, difficulty for finding and developing a successful controller for the maze decreases. This is compatible with the proposed definition of difficulty given in section 3.5. In fact, PLMan meets the requirements (stated in section 3.3) for an activity to be measurable with the proposed definition of difficulty:

- Progress in the activity is measurable. Each created controller gets an score depending on the percentage of eaten dots. Therefore, the progress is quantifiable and comparable as the score associated to the developed controllers.

- Score can be considered as a non-strictly increasing function over time. The only consideration to meet this requirement is that once a controller $C_i$ achieving a score $S_i$ has been developed, there is no way to get less than $S_i$ as score. Let’s imagine that a new controller $C_{i+1}$ is developed by modifying $C_i$. Then, if $S_{i+1} < S_i$, $S_i$ can be obtained again by restoring the controller $C_i$ and removing $C_{i+1}$. As controllers are Prolog programs stored in text files, saving and restoring is always possible, so achieved score is theoretically never lost.

- PLMan mazes have maximum score which is also associated with the success status. Success status is achieved when meeting the
goal of eating all the dots in a maze, thus getting 100% score.

• The activity is considered over time, and score is in fact measured at each step in the programming process. Each time a student has a new controller, it is tested and evaluated. If new controller improves previous controllers, score is updated with its new, higher value.

So, PLMan meets all the requirements for applying the proposed definition of difficulty. In fact, this analysis is also valid for many games that behave similar to PLMan. This means that the difficulty of many games can be modelled and measured using the proposed definition of difficulty. This also yields an interesting consequence: many games can be used as models to develop similar activities, in order to make designed activities ready to be modelled and measured with the proposed definition, with respect to its difficulty. Therefore, game mechanics is an interesting source of ideas for learning activities compatible with the results of this work.

4.5 Measuring difficulty of PLMan mazes

PLMan is one of the main assignments students have to complete in the subject Matemáticas 1 (Maths 1). This subject is taught in the Computer Engineering degree and the Multimedia Engineering degree at the University of Alicante. Students are asked to solve some mazes and they are given marks for each solved maze. The complete process is described in appendix A.

At the start of the 2014/2015 course, PLMan was modified to collect
information about the process that students followed to create controllers for mazes. Up to then, the information available came only from the final submissions that students made to the automatic assessment system that was specifically designed for the subject. The problem was that students were only submitting their controllers when they were finished, and there was no information about how much time had they invested in creating these controllers. After modifying PLMan, all controllers created by students are submitted along with their creation time. With these new information, it is possible to have much better estimations on the actual time invested by students in creating controllers for their assigned mazes. Moreover, submitted controllers are automatically evaluated, yielding information about students’ progress, as the system stores controllers associated to their score and creation time. This modification has been used to collect all progress from all the students who took part in the 2015/2016 course. All this data is used in this work to extract measures about the learning process students followed. This data includes progress and controllers created by 336 active students, and 220 available mazes.

The first important outcome from collected data is a measure for the actual difficulty of the mazes, according to the definition. Figure 4.7 shows functions $E_t$ and $D_t$ for mazes 1-00 and 1-09 that students face on their first stage with PLMan. Green line shows $E_t$ which represents the accumulated increase in score achieved by all the students that faced these mazes. So, $E_t = 1$ means all students achieved 100% score, and

---

4 This system is thoroughly described in appendix A
5 Data was conveniently anonymised first.
$t$ for that event is the time when the last controller getting that 100% score is submitted. Graphs show $t = 0.218$ hours ($\approx 13$ minutes) for all the students (20) to achieve 100% on the first maze and $t = 0.576$ hours ($\approx 34.5$ minutes) for the second one (14 students). $D_t$ is represented by the red area.

![Graphs showing $E_t$ and $D_t$ for PLMan maze 1-00 (up) and 1-09 (down).](image)

Figure 4.7: $E_t$ and $D_t$ measured for PLMan maze 1-00 (up) and 1-09 (down).

Although both mazes are classified as stage 1 - difficulty 1 by experts (professors), graphs show differences on their actual difficulty measures. Let us use $D_t$ for comparing both mazes $m = (1-00, 1-09)$ at times $t =$
At $t_1$, $D_{t_1}^{m_1} = 0.62 < 0.84 = D_{t_1}^{m_2}$, meaning that it was more difficult for students to start developing controllers for maze 1-09 than for maze 1-00. In fact, as $D_t$ takes into account past events on every $t$ measured, relation between 0.84 and 0.62 is not linear. So, learning curve is steeper for maze 1-09. At $t_2$, difference increases $D_{t_2}^{m_1} = 0.323 < 0.74 = D_{t_2}^{m_2}$, showing a clear difference on measured effort required by students. Moreover, arrived the time when both groups of students have completed 100% ($E_{t_2}^{m_1} = E_{t_3}^{m_2} = 1$), maze 1-09 still has $D_{t_3}^{m_2} = 0.4375 > 0.323 = D_{t_2}^{m_1}$. It follows that those who have faced maze 1-09 have realized a greater effort, according to $D_t$. This is a consequence of the definition of difficulty: its value collects and sums up all the history of the graph, and not only the score status at a given time. Therefore, comparing two values at a given time $t$, yields information about the effort required up to $t$. This is much more informative than using score or $E_t$, as it includes a hint on the slope of the learning curve up to the measured time.

Let us compare some more complex mazes. Figure 4.8 shows two mazes classified as stage 2 - difficulty 3. Maze 2-13 includes new objects and enemies: % represents pushable solid blocks, \ is a lever used to push % blocks, ) and ( are automatic archers that shoot arrows to Mr.PLMan when they see him, and F is a phantom that follows Mr.PLMan when seeing him. To be successful, controllers have to make Mr.PLMan get the lever, dodge arrows and the phantom, push the blocks (as there are dots down the blocks) and, of course, eat all the dots. This increased complexity requires more time for being tackled and that is clearly shown

---

6 Corresponding to (6, 12.6, 34.5) in minutes.
in the graph. Maze 2-32 is simpler with respect to items and enemies: \(E\) are patrolling enemies that move left-to-right, and \(\%\) is just a solid block that can be picked up by Mr.PLMan to clear the way. This conceptual simplicity is not correlated with a lower difficulty, as can be seen in figure 4.8.

Figure 4.8: \(E_t\) and \(D_t\) measured for PLMan maze 2-13 (up) and 2-32 (down).

Comparing mazes \(m = (2-13, 2-32)\) some similarities are apparent and some conclusions are obtained. Let us consider times \(t = (0.5, 4.07, 6.55)\). Both mazes show a fast start where most of the students develop controllers able to get most of the dots in a very short period of time. Al-
though being mazes that require greater effort to get 100% score, scores around 50%-75% are easier to achieve. That yields $D_{t_1}^{m_1} = 0.46 < 0.63 = D_{t_1}^{m_2}$, meaning that more dots are obtained by students’ first controllers on maze 2-13 than on maze 2-32. That justifies $D_{t_1}^{m_2} > D_{t_1}^{m_1}$, as students have more percentage of dots pending on maze 2-32 than their partners on maze 2-13.

Although progression in this mazes is similar, as $D_t$ curves show, it also shows an interesting difference in the tendency towards the end. The curve for maze 2-13 shows a tendency to decelerate its decreasing as time progresses. This continues with the previous analysis, as this maze has a difficult part that happens towards the end of the execution. $E_t^{m_1}$ curve shows this idea in a different way: it grows very fast in the interval [0, 0.5], but then it continues growing very slow. Students achieve a good controller very fast, but making that controller eat the last dots of the maze is much more difficult and takes the rest of the time. All these things happen in a similar way in maze 2-32, but with some interesting differences. The first part, where score grows fast, is much thinner. In fact, this part seems to last for a few minutes: the time required to develop a base controller that follows dots on orthogonal directions. This is learnt by students quickly on stages 0 and 1, and they are able to apply it in a few minutes on stage 2. These first controllers are comparatively poorer than the ones developed by their partners for the maze 2-13. However, students manage to present much more regular and continuous improvements to these controllers. This is shown by the steeper slope of $E_t^{m_2}$, and a much more maintained decreasing rate by $D_t^{m_2}$. 
In this case, 22 students were assigned maze 2-13; by the time $t_2 = 4.07$, all of them had achieved 100% score. However, students presented with maze 2-32 (28) were not so lucky. Most of them achieved 100% score, but some of them did not. Concretely, 3 of them did not manage to get 100%, being 77% the minimum score achieved in this maze. This explains why $E_{t_3}^{m_2} = 0.9592 < 1$. Nonetheless, final difficulty values (those obtained at the time for the last submission) are much similar this time: $D_{t_2}^{m_1} = 0.1905 < 0.248 = D_{t_3}^{m_2}$. But, even being this values more similar, it is important to take into account not only their difficulty value, but also the time difference: not only $D_{t_2}^{m_1} < D_{t_3}^{m_2}$, also $t_2 << t_3$, and even all controllers for $m_1$ resulted 100% successful.

These analyses show how the proposed definition of difficulty results in a powerful tool for comparing actual difficulty of learning activities (developing controllers for mazes, in this case). $D_t$ can be used to make generic comparisons of difficulty based on the area under the curve, slope and changes. But it also can be used for establishing comparisons based on concrete instants $t$. These possibilities have been shown in this chapter, demonstrating that some expert labelled difficulties did not have a good correlation with actual results got from students.
Chapter 5

Estimating Learning Difficulty with Neuroevolution

Previous chapters show a definition of difficulty (chapter 3) along with some measures taken for the PLMan game, which is used as learning activity (chapter 4). Measures taken using the proposed definition of difficulty show that it has interesting properties as a function that lets analysing actual difficulty and student progression. Although these are interesting results, the main question remains to be addressed: is it possible estimate the difficulty of a learning activity before handing it to students? And also, is it possible to automatically generate this estimation? Then, other questions about accuracy and validity of the estimations will
also arise.

These questions are related to the main claim of this work. This chapter shows a first approach on addressing these questions. Next sections show how to estimate the difficulty using Neuroevolution, and how to measure the accuracy of the estimations through a similarity measure. Several Neuroevolution methods are considered and two of them are directly applied and analysed.

5.1 How to estimate difficulty

Hausknecht et al. (2012, 2013) show that Neuroevolution has the potential to learn how to play Atari games directly from visual input. On their work, they develop several Neuroevolution architectures mainly using NEAT and HyperNEAT (Stanley, 2004, Stanley et al., 2009, Stanley and Miikkulainen, 2002d, 2004a) that learn to play different Atari games from scratch. Their developed systems use either visual preprocessed information (i.e. object classes and their location on the screen) or direct pixel values to feed neural networks. These neural networks output actions directly to a joystick controller (see figure 5.1), selecting next movement for controlled characters, as well as pressing fire button when required.

On analysing these previous works, one interesting idea comes to mind. These learning algorithms require time and generations to adjust and learn. Depending on the task given to these algorithms, number of generations and required time varies. This links with the intuitive
5.1. How to estimate difficulty

Figure 5.1: On the left, 4 examples of Atari games that Neuroevolution learns to play. On the right, an Atari joystick along with its 9-command representation (8 movement commands and 1 fire button).

notion of difficulty, and with this work’s proposed definition. Then, is there any relation between human-perceived difficulty and Neuroevolution’s learning cost? This interesting question links human learning costs to machine learning costs, assuming that both may be related. If that was the case, one could be used to estimate the other and vice-versa. Let’s ask it other way: would it be possible to establish a correlation between learning costs for a human learner and for a Neuroevolution algorithm?

To obtain a first answer to these questions, let us set up an empirical plan. Having actual results on how students progressed when learning to solve PLMan mazes, next step is getting same information from Neuroevolution algorithms. Therefore, systems able to learn to play PLMan will be required. Then, anticipating that measures will be different (human results are measured on required time, learning algorithms’ on required generations), a way to compare them will be needed. With all these tools set up, experiments will be run and results will show empiri-
cally if there is chance for a correlation or not.

5.2 Learning to play PLMan automatically

Let us start by designing a way to make Neuroevolution learn how to play and solve PLMan mazes. Following results in Hausknecht et al. (2013), NEAT and HyperNEAT algorithms seem possible right choices for this task. At a first glance, it should seem fair making a design that forces both algorithms to do the same task as students do. That is, developing Prolog Controllers to solve mazes. Such a design would require involving some kind of Genetic Programming (Buk et al., 2009, Lehman and Stanley, 2010, 2011). Outcomes from each learning algorithm should then be in the form of a Prolog program. However, it should be taken into account that the goal of these research is to find evidence of potential correlations between Neuroevolution and human’s cost on learning. Consequently, the most important thing on choosing a learning model is its expected effectiveness on learning to play PLMan. Whether the algorithm learns in a similar way from humans or not is not really important: the only important point is the potential correlation between learning costs.

Moreover, the Genetic Programming alternative would imply deeply modifying NEAT and HyperNEAT, because both algorithms work with neural networks and focus on producing “controller networks”: producing a Prolog program as outcome is a completely different task. This alternative should be considered a new research line in its own right, and
would be so different from reference works that results will be fully un-
predictable. On the contrary, using NEAT and HyperNEAT to produce
neural networks that directly control Mr.PLMan is a similar use to those
found in the literature. That means that previous research support the
idea that NEAT and HyperNEAT will be able to develop successful con-
trollers for Mr.PLMan. It also gives some guidance on how to apply and
tune the algorithms and what final performance is to be expected.

Therefore, in order to focus on the main goal of this research, NEAT
and HyperNEAT will be used to produce controller neural networks.
Firstly, that alternative offers some guarantees on a reasonable perfor-
mance learning to play PLMan. And, most importantly, there is no
scientific prove supporting the idea that learning costs will have a better
correlation to humans’ on any selected alternative. Investing the time
and effort to produce a new algorithm could even lead to worse corre-
lation results. Therefore, although it may seem intuitive selecting an
alternative that mimics the learning task humans do, it would deviates
this work from its main goal.

5.3 Designing the neural networks

This work’s proposal is similar to Hausknecht et al. (2013). Mr.PLMan’s
sensors are to be used as input for the neural networks. Also, a specific
design for using the complete maze as input will be mimicking the pixel-
by-pixel approach from Hausknecht et al. (2013). Approaches are similar,
but the one proposed requires adaptation, as input will not be pixels but
characters, which is similar to object classes. As for outputs, they will control Mr.PLMan’s actions directly, selecting one out of 16 possibilities.

Both NEAT and HyperNEAT train general neural networks: those including feedback links and loops. Both of them require their inputs and outputs to be designed. HyperNEAT also requires its internal substrates and inter-substrate connections to be designed. Basic input design for NEAT is shown at figure 5.2. Neural network is fed with inputs coming from Mr.PLMan’s short range sensor. In the input layer, a neural block is created for each cell from the sensor. Each neural block has as many input neurons as object classes available in the map. Neurons are binary: if the cell they are linked to contains the object class that the neuron represents, a 1 is input, otherwise a 0 enters the neural network.

![Figure 5.2: Input design using PLMan’s short range sensor for NEAT](image)

The basic input model proposed in figure 5.2 can also be easily extended to admit inputs from more sources. For instance, a 5x5 cell square
5.3. Designing the neural networks

could be used instead of the basic 3x3. Although being more complicated, an input model can also be designed for the list sensor. The list sensor has variable number of total inputs depending on Mr.PLMan’s location and the layout of solid objects in the maze. A simple solution is adding as many neural blocks as required for the maximum length of the list sensor for the maze. Then, for a 10x10-cells maze, 8 neural blocks would be required as input for each orthogonal direction, making a total of 32 neural blocks.

With respect to the output design, there are two main output designs that are shown at figures 5.3 and 5.4. First output design (figure 5.3) has one neuron layer with all possible individual actions that Mr.PLMan is able to perform. Each action in this model is in the form action(direction), as PLMan actions performed by Mr.PLMan are always related to an orthogonal direction of execution. For instance, Mr.PLMan can launch a ball to the left (use(left)), get a key that is one cell down (get(down)), or drop an object to the right (drop(right)). This is the easiest approach and has the advantage of mapping neural network directly to concrete decisions, not requiring a post-process step to convert them into final actions. However, this approach increases the number of nodes in the neural network: the final network is more complex and requires more time for training.

Second output design (figure 5.4) has been created with the aim of reducing the total number of required output nodes. The focus on this design is on optimization, without making it difficult for the network to correlate the learning outcome with the behaviours that produce it. So,
Figure 5.3: NEAT/HyperNEAT output design with explicit actions

Figure 5.4: NEAT/HyperNEAT output design split in actions and movements
two output neural blocks have been created: the first block behaves as an action selector (move, get, drop, use), and the second one as direction selector. This approach halves the number of output nodes (from 16 to 8) without making outputs more complex to interpret.

HyperNEAT has a different input model based on 2D substrates. Figure 5.5 shows an input design using 3x3 substrates, that will feed a HyperNEAT network using Mr.PLMan’s short range sensor (similar to figure 5.2 for NEAT). The size of each substrate can be 3x3, 5x5, etc. up to the size of the complete maze. As HyperNEAT is much less limited with respect to the number of neurons, it is wise to test it with higher input sizes. If the selected size is equal to the complete maze, the input-substrate layout will be different for each maze. For instance, maze 2-45 shown at figure 5.6 will require $19 \times 11$ neurons for each input substrate (one for each character from the maze).

HyperNEAT also requires its internal substrate architecture to be designed. This internal architecture corresponds to the hidden nodes of a general neural network, but disposed in 2D substrates. Selected proposal for this work includes 2 fully-connected, recursive, hidden substrates. This design aims to easily let the network construct patterns with 2 levels of abstraction: having $16 \times 10$ neurons in substrate 1 and half ($8 \times 5$) in substrate 2. Substrates are fully connected in forward sense (from input to output), without initial connections between nodes from the same substrate. Also, there are recurrent links from output layer to hidden layer 2 and from hidden layer 2 to hidden layer 1. The decreasing distribution of nodes in hidden layers also aims to force the construction
Figure 5.5: HyperNEAT input design. There is an input substrate for each type of object that can be found on the maze. The size of each input substrate is the same as the selected input grid. Selected grid is 3x3 in this example.

Figure 5.6: PLMan maze 2-45, classified as stage 2, difficulty 2 by professors.
5.4 Measuring learning difficulty for Neuroevolution

Next step is defining the way in which training cost will be measured. The logical way to take this measure consists in applying proposed definitions for $S_t$, $E_t$ and $D_t$. However, there is an important detail to take into account: $S_t$, $E_t$ and $D_t$ are relative to time $t$, which is a continuous value (usually measured in seconds). But measuring a Neuroevolution al-
algorithm in human time (i.e. seconds, hours, etc.) is not a valid measure in general. The algorithm may take different amounts of time depending on the computer where it is executed, on which other applications are executing in the same computer, on the way the algorithm has been implemented and even on the compiler used. As a Neuroevolution algorithm is based on a Genetic Algorithm, it is much more natural to measure time in training epochs passed. When a problem is more difficult to solve, a Genetic Algorithm requires more training epochs to succeed.

Measuring Neuroevolution’s learning cost in passed training epochs has a disadvantage. The number of passed epochs is a discrete value which makes it not directly comparable to other continuous values, like the human time passed. To address this issue, especial case definitions for $S_t, E_t$ and $D_t$ have to be defined. These definitions have to take into account the discrete nature of the number of training epochs.

Let us define $t'$ as the number of training epochs: the measure to be used as “time” in next especial case definitions. Let $\omega \in \Omega$ be a neural network controller from the set of neural network controllers produced by a Neuroevolution algorithm during training. Let $z \in Z$ be a maze from the set of all available PLMan mazes $Z$. Let $S(z, \omega)$ be the score that $\omega$ achieves when being run against maze $z$. This score will follow the general definition of score for the PLMan game\(^1\), but for a neural network controller instead of a Prolog controller.

At each epoch $t'$ there will be a population of neural network con-

\(^1\)The percentage of eaten dots during execution, minus corresponding punishments for each punishable event that happened. Score will never be greater than 100%, nor lower than 0%. More details on this in section 4.1
controllers $\Omega_t'$. This population will be evaluated by the Neuroevolution algorithm to assess fitness of all $\omega_t' \in \Omega_t'$. After this evaluation, there will be a best individual $\hat{\omega}_t'$ accomplishing this property:

$$\hat{\omega}_t' \geq \omega_t' \quad \forall \omega_t' \in \Omega_t'$$ (5.1)

That best individual $\hat{\omega}_t'$ will be the one used to measure progress of the Neuroevolution algorithm. This is logical, because having one neural network controller that achieves a concrete result is enough from the experimental point of view. As long as $\hat{\omega}_t'$ achieves a score $S(z, \hat{\omega}_t')$, it can be safely stated that Neuroevolution is able to achieve score $S(z, \hat{\omega}_t')$ after $t'$ training epochs.

Let $S^*(z)$ be the maximum score achievable for any possible controller in the maze $z$. $S^*(z)$ accomplishes this property:

$$S^*(z) \in \mathbb{R}, S^*(z) \geq S(z, \omega_t') \quad \forall \omega_t' \in \Omega_t'$$ (5.2)

These definitions help constructing a logical adaptation of $E_t$ and $D_t$ for Neuroevolution algorithms. Let us define $\dot{E}_t'(z)$ as the easiness function based on Neuroevolution-generated controllers applied to maze $z$ at training epoch $t'$. Equation 5.3 defines the value for $\dot{E}_t'(z)$.

$$\dot{E}_t'(z) = \frac{S(z, \hat{\omega}_t')}{S^*(z)}$$ (5.3)

Once $\dot{E}_t'(z)$ is defined, the definition of difficulty for Neuroevolution-generated controllers applied to maze $z$, $\dot{D}_t'(z)$ follows immediately. As equation 5.4 shows, the main difference between $D_t$ and $\dot{D}_t'(z)$ is the
discrete nature of $\dot{D}_{t'}(z)$, due to the use of training epochs $t'$ as time measure.

\[
\dot{D}_{t'}(z) = \begin{cases} 
0 & \text{if } t' = 0 \\
1 - \frac{1}{t'} \sum_{j=0}^{t'} \dot{E}_j(z) & \text{if } t' > 0 
\end{cases} \tag{5.4}
\]

In fact, discrete nature of $\dot{D}_{t'}(z)$ poses a problem, because $D_t$ is defined as a continuous function and comparing both of them is not straightforward. To solve this issue, let $\tilde{E}_t(z)$ be a continuous version of $\dot{E}_{t'}(z)$ assuming that learning between epochs is linear\(^2\). Equation 5.5 shows the concrete definition of $\tilde{E}_t(z)$, where $t$ is time measured in “continuous training epochs”.

\[
\tilde{E}_t(z) = \dot{E}_{\lfloor t \rfloor}(z) + (t - \lfloor t \rfloor)(\dot{E}_{\lfloor t + 1 \rfloor}(z) - \dot{E}_{\lfloor t \rfloor}(z)) \tag{5.5}
\]

Finally, equation 5.6 shows the definition of a continuous difficulty function across training epochs, which follows the continuous easiness function $\tilde{E}_t(z)$. Both functions use continuous training epochs $t$ as time, making them much more suitable for establishing useful comparisons with human-measured easiness and difficulty $E_t, D_t$.

\[
\tilde{D}_t(z) = 1 - \frac{1}{t} \int_1^t \tilde{E}_t(z)dt \tag{5.6}
\]

With this final definition for $\tilde{D}_t(z)$ a comparison with the measured difficulty for human learners will be easier to do.

\(^2\)This is the same assumption that is proposed in section 3.5 for human learners.
5.5 Similarities between humans and Neuroevolution

Thanks to the proposed definitions for $D_t$ and $\tilde{D}_t$, now it is possible to measure learning costs either for humans or for Neuroevolution algorithms. The next step is defining a similarity measure to be able to compare both costs. Once both measures are comparable, there exists the possibility to establish mathematical correlations that can finally lead to the main purpose of this work: being able to predict one cost through the other.

The similarity measure has to be a real value that gives a hint on how similar two given difficulty functions are. With the proposed difficulty definitions, this similarity can be defined generically. This means that the measure will be useful either for comparing measured $D_t$ for two mazes ($D_t(z_0) \approx D_t(z_1)$) or for comparing measured $D_t$ with estimated $\tilde{D}_t$ for the same maze ($D_t(z) \approx \tilde{D}_t(z)$). However, in order not to diverge from the main goal, let us start defining a similarity function focused on the connection between human and Neuroevolution learning costs.

Let $\sigma_t(z) \in [0, 1]$ be the similarity between measured human learning costs $D_t(z)$ and Neuroevolution learning costs $\tilde{D}_t(z)$ for the maze $z \in Z$. $\sigma_t(z) = 1$ means full similarity, which should only happen when both functions are the same function up to $t$. On the contrary, $\sigma_t(z) = 0$ means null similarity, that would represent maximum distance between all measured points from both functions, up to $t$. Summing up:
\[
\sigma_t(z) = 1 \text{ iff } D_j(z) = \tilde{D}_j(z) \quad \forall j \in [0, t] \\
\sigma_t(z) = 0 \text{ iff } D_j(z) - \tilde{D}_j(z) = 1 \quad \forall j \in [0, t]
\] (5.7)

With these design constraints in mind, it seems natural to define similarity as the complement to the area between both functions \(D_t\) and \(\tilde{D}_t\). This area is no other thing that their accumulated distance over time. This is a quite intuitive and natural definition, because functions become less similar when their instant distance \(D_t - \tilde{D}_t\) is greater, and also when distances are maintained over time. Figure 5.8 shows an example for better understanding this similarity measure.

![Figure 5.8: Graphically measuring similarity between \(D_t\) and \(\tilde{D}_t\) for a given maze \(z\). Area between the curves represents their distance (dissimilarity). The white area of the graph represents their similarity (complement to their distance).](image)

However, there is still one problem to solve in order to be able to
compare difficulties like in figure 5.8. Although time measure \( t \) for \( \tilde{D}_t \) is a continuous value, its range is different from that of \( D_t \). \( D_t \) is measured in hours, whereas \( \tilde{D}_t \) is measured in training epochs. To be able to compare both, a transformation is required. Let \( t \in [0, T] \) be the time measured in hours, and \( t' \in [0, T'] \) be the time measured in training epochs. Let us define a transformation \( \Theta(t) \) as follows:

\[
\Theta(t) = \frac{tT'}{T} = t'
\]  

(5.8)

By using scaling transformation \( \Theta(t) \), time-spaces for hours and training epochs become isomorphic. This makes the curves defined by \( D_t(z) \) and \( \tilde{D}_{t'}(z) \) lay into the same space, becoming comparable. Finally, this transformation can be applied to the notion of similarity, to define it mathematically. Equation 5.9 defines similarity as the complement to the area between \( D_{\Theta(t)} \) and \( \tilde{D}_{t'} \). This is now valid as similarity measure because both functions share the same space, after applying \( \Theta(t) \) transformation.

\[
\sigma_t(z) = 1 - \frac{1}{t} \int_0^t |E_t(z) - \tilde{E}_{\Theta(t)}(z)|dt
\]  

(5.9)

This final notion of similarity that is defined in equation 5.9 is simple, yet effective. It yields a value in \([0, 1]\) that tends to 1 when difficulties are similar, and tends to 0 otherwise. It is also important to state that \( \sigma_t(z) \) is related to a point in time \( t \), as all other measures defined in this work. This is a powerful concept for similarity analysis, because it lets us compare difficulties over time ranges. For instance, figure 5.8 shows
that $D_t$ and $\tilde{D}_t$ are very similar at their start, up to $t = 0.22$. From then on, difficulty differences start increasing up to $t = 0.6$. For the rest of the time, $D_t$ and $\tilde{D}_t$ become more and more similar to each other. This kind of analysis may yield important information about the nature of the activity being learnt. It also will point out differences in the way humans and algorithms deal with different problems. It is an interesting result as it points out to possible new research lines on learning.
Chapter 6

Results and Discussion

This work has proposed a definition of difficulty of learning activities as a function, an adaptation of this function to measure difficulty for humans, another adaptation obtain same type of measure for Neuroevolution algorithms and a definition of similarity between both measures. The main aim is to find a way to estimate difficulty of learning activities for students \textit{a priori}: previous to testing with real students.

This chapter focuses on testing these methods and definitions, for validating them against real results from students. These first tests constitute an initial empirical basis to estimate the validity of the whole approach. As this is a complete new line of research, it is wise to experiment with the concepts and obtain some empirical results. Then, if these results are promising, more research will be required to construct different experiments, test different learning activities, check different methods and, hopefully, induct some theoretical knowledge. To be practical, this
work conducts first experiments with proposed methods aiming for empirical validation.

To achieve the purposes of this work, an experimentation plan was designed, composed by these steps:

- Test NEAT and HyperNEAT algorithms against PLMan mazes. Set up all the environment to let a neural network controller give direct actions to Mr.PLMan. Then, launch learning experiments with both algorithms. Analyse results and early detect strong points and weaknesses.

- Select best input-output models and sets of parameters for learning. To do this, a set of mazes will be selected and learning tests will be conducted. The focus at this step is tuning NEAT and HyperNEAT up to the point they achieve the maximum possible score when playing the mazes.

- Finally, launch a training session with selected models and mazes, and compare learning results with results from real students.

### 6.1 Environment setup for Neuroevolution

In order to set up the environment for Neuroevolution, some interface design is required. Mainly, an interface for letting a neural network controller directly send commands to control Mr.PLMan, the same way that Prolog controllers do. This design is shown in figure 6.1. PLMan is implemented in Prolog language using SWI-Prolog (Wielemaker et al., 2012). Controllers that decide on Mr.PLMan’s actions are usually written in
6.1. Environment setup for Neuroevolution

Prolog language, and interact directly with the game through PLMan’s Application Programming Interface (API). PLMan was designed to operate this way for simplicity: anyone willing to create a controller for PLMan only needs a text editor and one first line to include API definitions. The rest is pure Prolog programming.

However, PLMan was not thought to be programmed in other languages. Then, as original NEAT and HyperNEAT implementations are in C++ language, a native code interface is required. For the purpose of this research, a native module for SWI-Prolog has been developed. This native module makes use of the native interface of SWI-Prolog to provide an interface to query NEAT and HyperNEAT neural networks from a Prolog program. By using this module, a simple Prolog controller can send input information to a neural network controller and get an output in the form of a concrete action to be performed by Mr.PLMan (see figure 6.1).

![Figure 6.1: Implementation of NEAT/HyperNEAT neural network controllers for PLMan](image)

Design shown in figure 6.1 also takes into account that neural network controllers require training. In this case, training is performed by a
genetic algorithm that works on top of NEAT/HyperNEAT algorithms. This genetic algorithm generates populations of neural networks that are stored as data files. Thanks to this, NEAT/HyperNEAT controllers have a unique general implementation, and their behaviour changes when different data files are used as input. This is the basis for the concrete training model implemented in this work, which is shown in figure 6.2. In this case, Controller encapsulates all the logic shown in the previous diagram (figure 6.1). This Controller module executes a PLMan game for each of the neural networks generated by the genetic algorithm. After each execution, a results file is generated with the achieved score and general statistics about the game. This results file is the input required by the genetic algorithm to evaluate the fitness of each individual of the population. Once fitness is evaluated for all individuals, an epoch is finished and a new population generated. This closes the training cycle, letting the genetic algorithm evolve the population of neural network controllers during as many epochs as required.

![Figure 6.2: Training model for NEAT/HyperNEAT controllers](image-url)
6.2 Fine tuning NEAT and HyperNEAT

NEAT and HyperNEAT are very powerful Neuroevolution algorithms that have shown the ability to succeed in a great variety of machine learning problems (Clune et al., 2009, Drchal et al., 2009, DAmbrosio et al., 2014, Gallego-Durán et al., 2013, Haasdijk et al., 2010, Hausknecht et al., 2012, 2013, Lee et al., 2013, Lowell et al., 2011b, Yosinski et al., 2011). However, this great potential comes at an important cost: both of them have plenty of free parameters to be set by researchers at design time. These parameters control different aspects of the way these algorithms work, and have a great impact on the final outcome. In fact, both NEAT and HyperNEAT usually show a rather low performance when applied out of the box to new problems. This lack of performance is generally due to an underperforming parameter configuration. For most problems, performance boosts when the appropriate parameters are set.

Unfortunately, there is no defined methodology or algorithmic way to find the optimal parameter set. The common way for setting parameters for this type of algorithms is by using expert knowledge about how these algorithms work, and a long trial-error cycle. Therefore, a lot of work has been carried out to find a fine tuned set of parameters. Concretely, the followed method to find a good set of parameters has been this one:

- Some mazes have been selected and classified with respect to their expert-assigned difficulty. These mazes will be used to measure the performance of NEAT and HyperNEAT given a concrete set of parameters.
• A score measure or meta-fitness function has been defined. This function evaluates the outcome of either NEAT or HyperNEAT given a set of parameters. The evaluations is done in terms of the score achieved on all the selected mazes and the required amount of effort to get the score.

• A concrete way to iterate through the parameter space has been selected. Taking into account that the parameter space is huge, a greedy approach has been selected. Although greedy approaches tend to stack on local minima, more sophisticated approaches have exponential cost with respect to the training time they will require. Therefore, a greedy approach may not be the best in terms of final result, but the final result will be available in a finite amount of time.

6.2.1 Selecting mazes for fine tuning

PLMan mazes are classified into 5 stages (0-4). Stage 0 is called “tutorial stage” as it only contains mazes that are solvable with 1 line of code. The intention is to introduce students to PLMan, showing them the actions Mr.PLMan can perform and how to code controllers and execute them. Therefore, these mazes are not useful from the Machine Learning perspective. All of them are discarded as neural network controllers will only have to learn to output one of the 16 possible actions, no matter what the maze contains.

At first, selected mazes were all mazes from stages 1 and 2. There are a total of 120 different mazes in stages 1 and 2. These mazes include
almost all the available mechanics in PLMan: corridors with dots and empty spaces, enemies, objects, doors, etc. Therefore, this is a perfect set in terms of knowing if an algorithm is fine tuned or not, looking at the scores achieved by the algorithm.

However, such a great number of mazes still proves to be prohibitive in terms of required training time, as it has to be multiplied by the amount of parameters configurations to test and the training iterations per set of parameters\(^1\). For instance, 67 different configurations are considered for NEAT, that are to be tested 2 times each, making a total of 134 configurations to test. With respect to HyperNEAT, there are 75 configurations, which makes a total of 150 configurations to be tested. That means \(134 + 150 = 284\) configurations. If that amount is to be tested on 120 different mazes, there is a total of 34080 tests to be run. Each test consists of 2000 generations of training, with an average time of 0.5 minutes per iteration. That would make a total of 65 years approximately. Even using 65 computers, a full year of calculations would be required just for selecting parameters using a greedy approach.

Therefore, a selection of mazes has been mandatory. Only 5 out of the 120 mazes have been selected for the fine tuning step. Mazes have been selected trying to include as many game mechanics as possible, trying to produce a representative selection. This makes an initial amount of required time of about 2.70 years. This amount has been reduced by using 2 computers and a parallel implementation, achieving a total required calculation time of 0.75 years approximately. Selected mazes are shown

\(^1\)NEAT and HyperNEAT parameters can be consulted in appendix B
in figure 6.3, whereas parameter ranges tested are shown in table 6.1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Tested set of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population Size</td>
<td>{25, 50, 100, 200, 400}</td>
</tr>
<tr>
<td>Elitism</td>
<td>{1, 2, 4}</td>
</tr>
<tr>
<td>Speciation</td>
<td>{true}</td>
</tr>
<tr>
<td>Crossover Rate</td>
<td>{0.10, 0.25, 0.50, 0.75, 0.90, 1.0}</td>
</tr>
<tr>
<td>Add New Loop Rate</td>
<td>{0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 0.96}</td>
</tr>
<tr>
<td>Add New Link Rate</td>
<td>{0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 0.96}</td>
</tr>
<tr>
<td>Add New Neuron Rate</td>
<td>{0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 0.96}</td>
</tr>
<tr>
<td>Weight Mutation Rate</td>
<td>{0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 0.96}</td>
</tr>
<tr>
<td>Weight Substitution Rate</td>
<td>{0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 0.96}</td>
</tr>
<tr>
<td>Weight Range</td>
<td>![Weight Range]</td>
</tr>
<tr>
<td>Survival Rate</td>
<td>{0.33, 0.50, 0.66, 0.80, 0.90, 1.0}</td>
</tr>
<tr>
<td>Maximum Stagnation</td>
<td>{10, 15, 25, 40}</td>
</tr>
<tr>
<td>Expressions Threshold</td>
<td>{0.05, 0.10, 0.25, 0.50}</td>
</tr>
<tr>
<td>Function Set</td>
<td>![Function Set]</td>
</tr>
<tr>
<td>Hidden Substrates</td>
<td>![Hidden Substrates]</td>
</tr>
<tr>
<td>Input Model</td>
<td>![Input Model]</td>
</tr>
<tr>
<td>Fitness Function</td>
<td>![Fitness Function]</td>
</tr>
</tbody>
</table>

Table 6.1: Candidate parameter sets tested using the proposed greedy approach

The selection of input model is included in the parameter selection, as shown in table 6.1. Although it is not to be considered a proper parameter from either NEAT or HyperNEAT, it is indeed a parameter from the proposed learning model. Therefore, it is included in the greedy approach for selecting the best input model along with the other parameters. The output model has been previously selected manually (see section 5.3) as both proposed options seemed logical, but the second one (8 outputs) has better performance because it saves some links to the neural networks, which means less parameters. This is much more important for HyperNEAT, because each removed output neuron saves from 50 to 1280 links,
depending on the selected hidden substrates model.

The parameter selection includes the fitness function, which could also be treated as a parameter, though it is not properly a parameter. In this case, the score achieved by a controller (as defined in section 5.4) represents a natural fitness function for evaluating its quality. As Neuroevolution aims to generate individuals achieving highest possible fitness, using score as fitness will make this process a way to generate controllers achieving maximum possible score.

Figure 6.3: Mazes selected to be used for finding the best set of parameters. These mazes include enemies (E), follower phantoms (F), keys (a), doors (l), levers (\), pushable blocks (%), automatic archers (V), laser shots (l), bombs (+) and guns (l).
6.2.2 Greedy approach for fine tuning

Let us call “a run of the experiment” to a full cycle of NEAT and Hyper-NEAT learning algorithms, completing 2000 iterations on each one of the 5 mazes with a concrete testing set of parameters. Using this definition, the greedy approach followed consisted in these steps:

- Establish a concrete set of parameters tuned by hand. These will be used as initial parameters for the greedy approach. These set is selected using some set of parameters found in the literature (DAmbrosio, 2011b, Stanley, 2004), and tuned minimally by testing and modifying them on some manual runs. Selected initial parameters are shown using *italics* in table 6.1.

- Within the initial set of selected parameters, all values are fixed except for the first parameter (Population size). Then, six runs of the experiment are performed for testing each of the proposed values for Population Size. After these six runs, the one getting the best global performance (as sum of the scores got on the 5 mazes) is used to fix the Population Size parameter.

- After fixing the Population Size, all parameters are fix again except the second one: Elitism. Similarly to Population Size, all 3 values of Elitism are tested on three runs of the experiment and the best result is selected as value for Elitism.

- This procedure is repeated for all parameters.

- Once all the parameters are fixed with their best individual values,
the full cycle is repeated for a second time, but starting with these
new fixed parameters. After this second refinement cycle, the set
of fixed parameters is selected as final.

Final selected parameters after performing this greedy algorithm are
shown on table 6.2. This selection is used for the next experiments.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>NEAT</th>
<th>HyperNEAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population Size</td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>Elitism</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Speciation</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Crossover Rate</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>Add New Loop Rate</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>Add New Link Rate</td>
<td>0.64</td>
<td>0.32</td>
</tr>
<tr>
<td>Add New Neuron Rate</td>
<td>0.16</td>
<td>0.08</td>
</tr>
<tr>
<td>Weight Mutation Rate</td>
<td>0.32</td>
<td>0.08</td>
</tr>
<tr>
<td>Weight Substitution Rate</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td>Weight Range</td>
<td>[-4,4]</td>
<td>[-4,4]</td>
</tr>
<tr>
<td>Survival Rate</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td>Maximum Stagnation</td>
<td>15</td>
<td>25</td>
</tr>
<tr>
<td>Expressions Threshold</td>
<td>N/A</td>
<td>0.25</td>
</tr>
<tr>
<td>Function Set</td>
<td>N/A</td>
<td>{ Gaussian, Sigmoid,</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
<td>Sine, Absolute Value }</td>
</tr>
<tr>
<td>Hidden Substrates</td>
<td>N/A</td>
<td>(16x10, 8x5)</td>
</tr>
<tr>
<td>Input Model</td>
<td>3x3</td>
<td>Full Maze</td>
</tr>
<tr>
<td>Fitness Function</td>
<td>$S(z, \omega_t)$</td>
<td>$S(z, \omega_{t''})$</td>
</tr>
</tbody>
</table>

Table 6.2: Selected parameters for NEAT and HyperNEAT after testing
the set of candidate parameters using the proposed greedy approach.

6.3 Experimental results

After setting up all the environment properly, the last step is launching
the experiments designed to compare learning results from NEAT and
HyperNEAT to those achieved by students. For this purpose, all 120
mazes from stages 1 and 2 are considered. Mazes from stage 0 were discarded because of their simplicity: they are introductory and they will not yield much information. Mazes from stages 3 and 4 were discarded because of their added complexity: these mazes are non-deterministic, requiring statistical testing for all of them. These complexity is beyond the scope of this study and would make final results much more difficult to analyse. Therefore, as this is a first empirical validation, maintaining it simpler will yield a better perspective on the validity of the results.

Next results presented come from running NEAT and HyperNEAT up to 2000 training epochs for each maze using parameters from table 6.2. Execution was also stopped whenever 100% score was achieved. Considered score is always the score of the best performing individual $\hat{\omega}_t$ from each epoch $t'$. To maintain the selected linear model for $\tilde{E}_t$, only epochs that produce individuals better than previous best performers are considered. For instance, for a given maze $z$, let us consider the general learning scenario proposed by equation 6.1, with a sequence of best performing individuals. All individuals $\{\hat{\omega}_i, x < i < n\}$ are removed from the final results as they represent no improvement over $\hat{\omega}_x$. Then, linear interpolation is used between $S(z, \hat{\omega}_x)$ and $S(z, \hat{\omega}_{x+n})$, exactly the same as for student results.

$$\tilde{\omega}_{t'} \in \{x, x + 1, x + 2, ..., x + n\} \quad x, n \in \mathbb{N}$$

$$S(z, \hat{\omega}_{x+i}) = S(z, \hat{\omega}_{x+j})$$

$$S(z, \hat{\omega}_{x+n}) > S(z, \hat{\omega}_x) \quad \forall i, j \in \mathbb{N} / i < n \land j < n$$

(6.1)
Complete results from all the experiments can be found in table C.1 (in appendix C). Results are left there for reference, in order to make this chapter clearer and easier to follow. In this respect, two representative mazes have been chosen as example to analyse and discuss here. The first one is maze 1-41 (figure 6.4). This maze is classified as stage 1, difficulty 3 by professors. As difficulty goes from 1 to 5, this maze is a representative of the mean difficulty in the first stage that students have to pass. The maze contains a door |, a key a to open the door and 3 enemies E (two that move up-to-down and one that moves forming a square in front of the door). A successful controller for this maze has to dodge the enemies, get the key, open the door and eat 43 dots.

\[ \tilde{E}_{\Theta(t)}(1-41) \] and \[ \tilde{D}_{\Theta(t)}(1-41) \] are represented in the 2 upper graphs from figure 6.4. That is the representation of the evolution of difficulty for NEAT and HyperNEAT when learning to solve this maze. The two graphs at the bottom represent the distance of NEAT and HyperNEAT with respect to measured students’ difficulty \( D_t(1-41) \). As mentioned before, similarity is the complement to distance, so the lesser the distance the better the result as estimation for the actual students’ difficulty. For this maze, NEAT gives a relatively useful estimation for the actual difficulty experienced by students. Although similarity is not very high (\( \sigma_{2,8}^{\text{NEAT}}(1-41) = 0.874 \)), the graphical representation is useful as a mid-range estimation and for comparing it with other mazes for relative difficulty information. But HyperNEAT clearly yields a much accurate estimation for the actual difficulty experienced by students. Although similarity is not very high (\( \sigma_{2,8}^{\text{HyperNEAT}}(1-41) = 0.874 \)), the graphical representation is useful as a mid-range estimation and for comparing it with other mazes for relative difficulty information. But HyperNEAT clearly yields a much accurate estimation for the actual difficulty experienced by students. Although similarity is not very high (\( \sigma_{2,8}^{\text{HyperNEAT}}(1-41) = 0.874 \)), the graphical representation is useful as a mid-range estimation and for comparing it with other mazes for relative difficulty information. But HyperNEAT clearly yields a much accurate estimation for the actual difficulty experienced by students. Although similarity is not very high (\( \sigma_{2,8}^{\text{HyperNEAT}}(1-41) = 0.874 \)), the graphical representation is useful as a mid-range estimation and for comparing it with other mazes for relative difficulty information. But HyperNEAT clearly yields a much accurate estimation for the actual difficulty experienced by students. Although similarity is not very high (\( \sigma_{2,8}^{\text{HyperNEAT}}(1-41) = 0.874 \)), the graphical representation is useful as a mid-range estimation and for comparing it with other mazes for relative difficulty information. But HyperNEAT clearly yields a much accurate estimation for the actual difficulty experienced by students. Although similarity is not very high (\( \sigma_{2,8}^{\text{HyperNEAT}}(1-41) = 0.874 \)), the graphical representation is useful as a mid-range estimation and for comparing it with other mazes for relative difficulty information. But HyperNEAT clearly yields a much accurate estimation for the actual difficulty experienced by students. Although similarity is not very high (\( \sigma_{2,8}^{\text{HyperNEAT}}(1-41) = 0.874 \)), the graphical representation is useful as a mid-range estimation and for comparing it with other mazes for relative difficulty information. But HyperNEAT clearly yields a much accurate estimation for the actual difficulty experienced by students. Although similarity is not very high (\( \sigma_{2,8}^{\text{HyperNEAT}}(1-41) = 0.874 \)), the graphical representation is useful as a mid-range estimation and for comparing it with other mazes for relative difficulty information. But HyperNEAT clearly yields a much accurate estimation for the actual difficulty experienced by students. Although similarity is not very high (\( \sigma_{2,8}^{\text{HyperNEAT}}(1-41) = 0.874 \)), the graphical representation is useful as a mid-range estimation and for comparing it with other mazes for relative difficulty information. But HyperNEAT clearly yields a much accurate estimation for the actual difficulty experienced by students. Although similarity is not very high (\( \sigma_{2,8}^{\text{HyperNEAT}}(1-41) = 0.874 \)), the graphical representation is useful as a mid-range estimation and for comparing it with other mazes for relative difficulty information. But HyperNEAT clearly yields a much accurate estimation for the actual difficulty experienced by students. Although similarity is not very high (\( \sigma_{2,8}^{\text{HyperNEAT}}(1-41) = 0.874 \)), the graphical representation is useful as a mid-range estimation and for comparing it with other mazes for relative difficulty information. But HyperNEAT clearly yields a much accurate
Figure 6.4: Difficulty and similarity results for NEAT and HyperNEAT on maze 1-41 (Stage 1, difficulty 3). $D_t$ is the difficulty function measured for students that were assigned this maze. $\tilde{E}_{t'}$, $\tilde{D}_{t'}$, $\tilde{E}_{t''}$, $\tilde{D}_{t''}$ are the easiness and difficulty functions measured for NEAT and HyperNEAT. Distance is the graphical representation of $1 - \sigma_t^A$, for $A \in \{\text{NEAT, HyperNEAT}\}$. 
6.3. Experimental results

Figure 6.5: Difficulty and similarity results for NEAT and HyperNEAT on maze 2-11 (Stage 2, difficulty 5). $D_t$ is the difficulty function measured for students that were assigned this maze. $E_{t'}$, $\tilde{E}_{t'}$, $D_{t'}$ and $\tilde{D}_{t'}$, $E_{t''}$, $\tilde{E}_{t''}$, $D_{t''}$ are the easiness and difficulty functions measured for NEAT and HyperNEAT. Distance is the graphical representation of $1 - \sigma^A_t$, for $A \in \{\text{NEAT, HyperNEAT}\}$. 
estimation, with a similarity \( \sigma_{2.8}^{\text{Hyper}}(1-41) = 0.972 \). Graphical comparison between students and HyperNEAT shows the quality of this output as prediction for the actual difficulty.

As a representative for a more difficult maze from stage 2, figure 6.5 shows maze 2-11, which is classified as difficulty 5 by professors. This maze has 4 patrolling enemies \( E \) with defined squared routes, 2 teleporting stations \( ? \) that teleport Mr.PLMan from one side to the other and viceversa, 4 automatic archers \( ) \) (that shoot arrows \( > < \) when they see Mr.PLMan and a gun \( l \) loaded with 1 bullet that can kill an enemy. A successful controller for this maze has to enter the teleporter station and eat all the dots guarded by the automatic archers, and return to the main area to eat the rest of the dots, up to the total of 181 dots.

Results for maze 2-11 (figure 6.5) are worse than the ones obtained for maze 1-41 (figure 6.4). Again, HyperNEAT shows a much accurate prediction for actual students’ difficulty \( \sigma_{16.5}^{\text{Hyper}}(1-41) = 0.865 > \sigma_{16.5}^{\text{NEAT}}(1-41) = 0.737 \). But this time, difficulty prediction is not to be considered accurate enough, as a similarity of 86.5% equals to a difference of 13.5%, which may be too high depending on the context. However, graphical representation of the results continues to be very interesting. Both Neuroevolution algorithms show interesting behaviour similarities with respect to students. For instance, both show that delivering a controller able to eat \( \approx 50\% \) of the dots is relatively easy and fast to do. This is something students also show in their progress graph. It is important to take into account that students do not start from scratch, as they have already completed from 8 to 10 mazes when they face maze 2-11. Most
of them have already developed controllers for previous mazes that they cut and paste to obtain some initial good results with a minimum effort. That is the reason for the curvature of their difficulty graph. On the contrary, Neuroevolution starts from scratch each time, which represents a handicap. Therefore, seeing the similarity of the curvatures in the light of these facts yields a truly interesting outcome and poses more questions. Would the outputs had been more similar if students had started from scratch or, at least, with less previous experience?

6.4 Discussion on global results

NEAT and HyperNEAT show different abilities and ways of finding neural network controllers to solve PLMan mazes. In fact, the configuration selected for NEAT demonstrates being generally faster in finding valid neural network controllers (i.e. it takes far less epochs). However, it seems to have great tendency to get stuck to some concrete solutions, which is similar as being trapped on local minima. HyperNEAT shows slower pace on finding good controllers, but with better ability to get out of local minima. These differences between NEAT and HyperNEAT result crucial for the purpose of this work, because thanks to them results are better than expected. HyperNEAT’s slower pace correlates much better with students’ pace, setting it as a fairly good estimator for students’ learning difficulty. Complete quantitative results are shown in table C.1 for reference and further analysis.

The most interesting question posed at the start of this work was
about the existence of a correlation between Neuroevolution and students with respect to their learning progression when facing the same learning activity. The existence of this correlation would lead to comparable difficulty functions, making those generated with Neuroevolution valid for estimation purposes. To answer this question, the most important outcome is the distribution of similarities calculated for all the tested mazes. Therefore, considering $Z$ as the set containing all mazes in which NEAT and HyperNEAT have been tested ($|Z| = 120$), similarities are distributed as follows:

$$
\sigma_t^{\text{NEAT}}(z) \sim \mathcal{N}(0.845, 0.130) \quad \forall z \in Z
$$

$$
\sigma_t^{\text{Hyper}}(z) \sim \mathcal{N}(0.885, 0.126) \quad \forall z \in Z
$$

(6.2)

Figure 6.6 shows these distributions clearly. This confirms that there is indeed a correlation between NEAT / HyperNEAT difficulty and student difficulty to solve a PLMan maze. If that correlation did not exist, results would have been distributed uniformly, yielding maximum entropy. Being both similarities distributed normal has a practical meaning: given a random maze $z \in Z$, and the results of training NEAT or HyperNEAT to solve $z$, the most probable events are $\sigma_t^{\text{NEAT}}(z) = 0.845$ and $\sigma_t^{\text{Hyper}}(z) = 0.885$. Therefore, after training NEAT or HyperNEAT and calculating their difficulty function for a given maze $z$, it is wise to assume that the obtained difficulty function will have a similarity of 0.845/0.885, depending on the selected algorithm. This result may be used to create a predictor out of NEAT or HyperNEAT, establishing a confidence interval over results obtained from training.
Figure 6.6: Distribution of similarity for NEAT and HyperNEAT

The most important consideration about these results is that they are a first empirical validation for the proposed approach. It is important to be cautious, as these research has many limitations and is focused on one concrete activity (the PLMan game) and with a relatively small subset of mazes (120) and students (336). Results are significant in the sense that they are a first positive step for more research in the field. They confirm that for this concrete case and subset the correlation exists, which is certainly valuable. But generalization is still not possible: more research is required to show if general theoretical results can be obtained.

Another interesting consequence of the whole system designed resides in the difficulty graphs as valuable analysis tool. As it has been shown during previous analyses, even when similarity is low for a specific maze, the curvatures of the difficulty function transmit plenty of information
about the maze. Curvatures show easy and complicated parts of the maze, potential problems as well as keys about the relation between Neuroevolution’s behaviour and students’, on their way to find a successful controller. Section C.2 provides additional examples along with their analyses to further show the importance of this outcome.

It is also interesting to mention a couple of special cases. As can be seen on tables C.3 and C.4, there are two mazes without a similarity value: 2-06 and 2-31. Let us have a look at these two mazes to analyse the source of this issue (see figure 6.7). Both mazes have a similar structure: dots are hidden behind a mechanism. Maze 2-06 has a goal (□) that can only be destroyed by throwing balls (ο) and scoring 3 times. Balls are constantly appearing 2 cells away from the scoreboard. Maze 2-31 has a door (□) that has to be opened with a key (&). In both cases, the problem is the same for Neuroevolution: to solve the maze some concepts have to be understood and an action plan is required. Moreover, there are no dots that can guide to a better controller through an increment in fitness. Therefore, even if some individuals do partially perform some required actions, their fitness will always be 0, same as individuals that do nothing. This halts evolution, impeding learning through Neuroevolution’s genetic algorithms. Treatment for this special cases has been left out of the scope of this work, and is matter of discussion for future research.

Results presented confirm that the way in which tested Neuroevolution algorithms learn to solve a PLMan maze is correlated to that of students. This is a very important result, in the sense that it confirms that there are similarities in both ways of learning. Consequently, Neu-
Figure 6.7: Mazes 2-06 (left) and 2-31 (right). Neither NEAT nor Hyper-NEAT found a controller able to eat a single dot after 2000 iterations.

roevolution could be used to predict the difficulty of learning activities at design stages, previous to any test made with actual students.

However, it is also important to highlight the limitations of these results. Experiments carried out are only an empirical confirmation, so they only apply to the used dataset. Although these results seem to be probable on other similar learning activities, results from this work do not constitute a proof for that. As no theoretical proof has been carried out, another set of similar experiments should be run for different datasets to have empirical proofs that validate this approach particularly. Moreover, although results may be used as direct predictions, their accuracy is to be considered with care. Most probably, a direct use of the results as predictions may have a limited utility in an actual scenario. As stated before, a confidence interval is much recommended and, considering standard deviations from both distributions, that interval should have a considerable size. Depending on the application scenario, that may pose a problem.

Finally, results presented in this section validate the formal hypothesis
for this work\textsuperscript{3}. Although limitations may lead these results not to be useful under some practical environments, they represent a first step in this field. Future research on this topic may find other learning algorithms or training methods with better similarity results. This research lays the groundwork for future improved solutions that will be useful on any learning scenario.

\textsuperscript{3}stated in section 1.1
Chapter 7

Conclusions and Further Work

This work started by hypothesizing the possibility of using the training cost for Machine Learning algorithms to estimate the learning cost for humans. More precisely, it was questioned if new definitions for difficulty could be find, that could be measured from actual collected data and estimated using Machine Learning.

7.1 Contributions in this work

Following this hypothesis, a general definition for difficulty has been presented. This new definition has been designed on the bases of a list of desired properties. By using this proposed definition, difficulty becomes measurable, can be compared and visualized, and is related to effort over
time. Effort is modelled as required time to achieve a specific score value. Therefore, proposed definition of difficulty takes into account progress towards solving a learning activity, based on the score an agent achieves when performing the activity.

This proposed definition of difficulty has limitations in the sense that activities have to meet some requirements to be measurable under this definition. The activity should be performed over time\(^1\) and a score function to measure progress should be available. The score function should have upper and lower boundaries and be non-strictly increasing: there should be no possibility of losing score over time.

But this proposed definition has also many interesting advantages. Most of its advantages come from being drawable: this confers it the ability to show its progress over time, so that it transmits characteristics of the learner and the learning activity graphically. Different parts of the learning activity can be identified: for instance, most difficult parts will produce valleys in the graph that will permit not only their identification, but also their measurement. Activities can be compared using their difficulty graphs, yielding a much accurate knowledge about which ones require more effort, and the differences in the distribution of the effort over time. These advantages make the proposed definition of difficulty a powerful tool for analysing and comparing learning activities.

As specific activity to be used for empirical experimentation, a game called PLMan has been presented. The game is currently being used at

\(^1\)That is, not being defined as an instant activity. An example of an instant activity would be giving an answer to a 4-options question in paper, considering only the given answer and not the required time to think and produce the answer.
the University of Alicante to teach Prolog programming, Logics and a light introduction to Artificial Intelligence. PLMan has been shown to meet required properties to apply the definition of difficulty. It also has shown its potential as a generic class of learning activities: students solve mazes and each maze is a learning activity in its own. Therefore, PLMan is a generic class of learning activities with similar content but different difficulties.

It also has been stated that many games have an important parallelism with PLMan. This characteristic makes all these games adaptable to the proposed definition of difficulty. Therefore, they are to be considered as learning activities that could be measured and estimated by using the methods proposed in this research.

An specifically adapted version of the difficulty function has been created for PLMan. This version has been used to measure difficulty for 220 different mazes that have been performed by 336 students\(^2\). Some results of these measures have been presented and explained.

To further proceed with the goals of this research, Neuroevolution has been selected as learning method to automatically solve PLMan mazes. After reviewing the state of the art in the field, two specific algorithms have been selected: NEAT and HyperNEAT. Then, specific models for neural networks have been designed. Concretely, several input and output models have been proposed for NEAT and HyperNEAT, and various hidden-substrate models for HyperNEAT.

\(^2\)Each student was assigned some random mazes from the total available. Depending on their performance, each student end up having between 8 and 18 mazes assigned.
Moreover, an adapted definition of difficulty has been proposed to obtain measures for NEAT and HyperNEAT. This measure has to be compared to those obtained from actual students’ performance data. Therefore, a similarity function has been designed on the basis of both difficulty functions. This similarity function measures the complement to the area between both difficulty curves. This area measures the difference between the functions, so its complement measures the similarity, as it was desired.

An adaptation to PLMan has been developed in order to perform learning experiments with Neuroevolution, obtain its learning costs and measure its similarity with students. This adaptation lets Mr.PLMan be controlled by a neural network instead of a Prolog knowledge base. After that, NEAT and HyperNEAT have been fine tuned: sets of configuration parameters have been tested following a greedy approach, and those with best performance have been selected.

With the appropriate parameters selected, NEAT and HyperNEAT have been trained to solve 120 PLMan mazes from stages 1 and 2. Results have given a measure of the learning cost for NEAT and HyperNEAT on solving these mazes. These results have been compared with those from students by means of the previously defined similarity function. This final comparison has yielded 240 similarity values: 120 related to NEAT learning costs versus students’ learning costs, and other 120 related to HyperNEAT versus students. NEAT similarities are distributed normal with mean 0.845 and standard deviation 0.130. HyperNEAT similarities are also distributed normal with mean 0.885 and standard deviation
0.126. This results have confirmed the initial hypothesis of this work: there is indeed a correlation between learning costs for NEAT / HyperNEAT and those for students within the context of the experiments and the data used.

Besides the validation of the main hypothesis, all contributions presented in this work are summed up as follows:

- A new definition of difficulty with interesting properties for analysing student progress on solving learning activities.
- The PLMan game as a learning activity and as a model for easily adapting other learning activities to meet the requirements of the proposed definition of difficulty.
- An application of Neuroevolution for learning to solve PLMan mazes automatically.
- A similarity function to measure estimation accuracy when using Neuroevolution learning costs as a prediction of students’ learning costs.
- A novel application of Neuroevolution to estimate difficulty of learning activities at design stages.

Furthermore, the main hypothesis has been validated by showing that there is a correlation between Neuroevolution learning costs and those of students. Therefore, students’ difficulty on solving a PLMan maze could be estimated with the difficulty function that results of measuring NEAT or HyperNEAT training to solve the maze. To be fair, it is important
to state that the accuracy of the estimation is not really high. However, using an appropriate confidence interval, estimation is valid as a first insight on the future cost students will have to invest on solving the maze. Moreover, comparison between different difficulty functions resulting from Neuroevolution training can also be used as estimative comparisons of actual difficulties for students. This last application would probably be much more accurate than absolute difficulty values.

The novel contribution presented in this work opens up a new field of research: using Machine Learning as a proxy for human learning.

7.2 Further work

As mentioned in previous section, one of the most interesting outcomes of this work is opening a new field of research related to the learning itself, the different ways of learning (humans versus Machine Learning) and existing correlations. This field may yield much interesting results in future research, as it has potential to discover new findings on the way learning happens on humans. Moreover, the feedback loop it generates also has potential to drive the development of new Machine Learning algorithms and improve existing ones on the basis of their similarity with human learning. These are interesting lines that will be challenging to explore but extremely rewarding if any of these foresights materializes.

However, these possibilities will probably require many previous steps and years of research. Being practical, these are next immediate paths that this research should follow:
7.2. Further work

- Develop a new way of training NEAT / HyperNEAT with the aim of getting the closer performance to humans’. In this work, both algorithms have been trained in the traditional way, trying to achieve the maximum performance in the sense of ability to solve PLMan mazes. However, could similarity functions be used as fitness function in any way? Is there any other way to train both algorithms to solve PLMan mazes as closer to the way humans do as possible?

- Export this method to other learning activities. Finding similar correlations on learning activities different than PLMan could confirm a more general pattern and may lead to develop theoretical results. Many different activities and tests are required.

- Propose initial theoretical hypothesis on the existence of a general correlation between Machine Learning and human learning. Could there be a general similarity between all possible learning models? This line poses a very long term work to achieve a goal, but the final goal could be highly valuable either in Artificial Intelligence or in social sciences (any field related with human learning or learning in general). Although it is early to start this line of research, it is important to take it into consideration, as often theoretical ideas feed practical experiments and generate useful feedback loops.
Chapter 7. Conclusions and Further Work
Appendix A

The PLMan Learning System

A.1 Structure of the PLMan Learning System

The PLMan Learning System is a custom-made gamified, automated learning system that gives support to a first-year subject whose aim is to introduce students into Computational Logic. This platform is structured around a gamified Website that manages all the information and elements of the system and allows the interaction of the actors (students and teachers). This is the entrance point for the students, who can download the mazes of the PLMan game, upload their solutions, obtain their marks and receive the predictions of the system about their learning progress.
The teachers, on their behalf, enter the system to manage the students’ activity, introduce new mazes, assign the initial stage and difficulty level of the mazes (notice that here is where the results of this research are to operate) and monitor the students’ progress.

The PLMan Learning System has recently improved by adding a predictive module that allows the interaction events to be registered in a database so that they can be used in further analysis. From these generated events and the students’ learning results, a set of representative features are extracted and used to predict the learning results of the students, using a Support Vector Machine (SVM) (Chang and Lin, 2011, Cortes and Vapnik, 1995) to analyse the data and classify the students according to their learning progress. The predictions are offered to teachers and students as a means to early detect learning dysfunctions.

The elements of the PLMan Learning System are illustrated in figure A.1, where a cyclic interaction pattern can be identified: students access the gamified platform where their actions and learning progress take place, their grades are accumulated, the generated events are registered, their learning results and the use data are processed and finally a prediction is made and offered to both student and teacher. This cycle is repeated for every stage of the learning process. In the following sections each element is explained in detail.
A.2. The gamified Website

The PLMan Website is the entrance site for every user. It includes a public and a private site. The public site just includes a welcome text as well as some instructions about the use of the system and some links to download the PLMan game. The private site requires authentication and provides the functionality for every user, given his or her particular profile.

Students have a general vision about their progress in solving the mazes and their accumulated grade. Figure A.2 shows the profile of a student at a given instant. Teachers, on their part, have an administration profile that lets them manage the students and groups, monitor the students progress, as well as uploading new mazes and assigning them their corresponding stage, difficulty level and solving deadline.
Appendix A. The PLMan Learning System

Figure A.2: Students’ view of the gamified Website at a given instant.

A.3 Practice exercises

Practice exercises consist of solving PLMan mazes as it was explained in chapter 4. More than 400 different mazes have been made for PLMan, with different layouts, objects to get and use, enemies and obstacles to avoid and even problems to solve. These mazes are organized into 4 main stages and up to 5 levels of difficulty per stage. All of them have been included into the PLMan Learning System.

Students have to beat the 4 stages and a checkpoint in the system to get the maximum grade (see table A.1). At each stage, students have to solve 1 to 5 different mazes. To get each new maze for solving, students start by pick up their desired difficulty level among valid levels for the stage in which they are. Then, the system presents them with a random maze from their selected level of difficulty. Mazes can only be assigned
once, so the same maze cannot be assigned to two different students.

Students use PLMan software to develop and test their solutions to each maze they get assigned. As previously discussed, solutions are in the form of a Prolog program that contains rules to guide the Pac-Man character. Every time they have developed a solution they consider to be working, they submit the solution through the PLMan Website. The automated system runs the solution and evaluates their score and marks based on the total proportion of dots their pac-man achieves to eat, minus some concrete punishments for errors made by their programs (like trying to move into a wall, for instance). The system shows detailed evaluation to the students and, if they complete more than 75\% of the maze, the unlock the next maze and can continue. If not, they have to improve their solution and send it again until they achieve 75\% or more. There is no limit on the number of solutions they can send, and they are not penalized for sending new solutions: they are only limited by stage deadlines that they have to meet.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Mazes</th>
<th>Difficulties</th>
<th>Marks</th>
<th>Deadline (week)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (tutorial)</td>
<td>5</td>
<td>1</td>
<td>1%</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1-5</td>
<td>5%</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1-5</td>
<td>7.5%</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1-5</td>
<td>17.5%</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2-4</td>
<td>22.5%</td>
<td>10</td>
</tr>
<tr>
<td>5 (checkpoint)</td>
<td>1</td>
<td>1-5</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table A.1: Structure of stages for mazes to be solved by students in the automated system

The system is designed in the aim of formative assessment, considering that students need to learn from their own mistakes without being
penalized for it. Because of this, students do not have a limit of submissions for a given maze. The system always consider the best solution they have submitted to give them marks. Partially solved mazes also contribute to their final grade proportionally to the percentage solved of the maze. Students can also follow their own path by selecting difficulty levels that make them feel more comfortable: the greater the difficulty level, the more the contribution to the final grade (Marks shown in table A.1 represent full marks per each individual average-difficulty map). They may also stop whenever they wanted: i.e. if they got to the 3rd stage and have accumulated 65% marks, they may stop solving mazes if they wanted and stay with that marks.

A.4 Prediction module

While designing, building, redesigning and maintaining a system like this, it becomes clear that the flow of data that goes in and out the system is invaluable from the teaching/learning point of view. The PLMan Prediction Module transforms these data into valuable information that can be used by teachers for advice about the progress of the students towards success in the early stages of the academic term.

Every week, during the interaction between the students and the system, every event is logged in an event database (for instance, show frontpage, select difficulty, download maze, submit solution, and so on) with their appropriate time-stamp and related information. Furthermore, several learning results from the students’ activity are also collected (for
A.4. Prediction module

Figure A.3: Evolutive probabilistic graph for a great student. Green color represents the probability of achieving great marks, blue the probability of normal marks, and red the probability of underperforming marks.

instance, percentage obtained $> 75\%$, time to solve each maze, and so on). All this set of data feeds a prediction algorithm, based on a Support Vector Machine, that classifies the expected student performance in 1 out of 3 possible classes, depending on the expected final marks out of 10 points ($em$): (1) Great students ($em > 8.05$), (2) Normal students ($8.05 > em > 5.75$) and (3) Underperforming students ($em < 5.75$).

The output of the prediction module is a weekly evolutive probabilistic graph like the one of figure A.3. These graphs show the estimated probabilities that each student has of becoming great (green/top), normal (blue/center) or underperforming (red/bottom). The probabilities are calculated each week and accumulated in the graph, showing student trends. This trend information is particularly useful for teachers to guide the students in their further performance.
A.5 Features of the PLMan Learning System

There are some fundamental features that characterize the PLMan Learning System, depending on the facet that is considered. As an interactive online system the main features are:

- High Availability, except in case of server failure.
- Intuitive interaction for students and teachers.
- Automation, so that it avoids the teacher overload.
- Objectivity, not depending on teacher’s evaluation.
- Equality for every student, who plays under the same conditions.
- Continuous assessment of the student’s work.

As a gamified system, it has the following characteristics:

- Simplicity, so that the initial objectives are affordable and stimulating, increasing complexity gradually.
- Feedback about the results, provided immediately.
- Real-time, both for interaction and assessment.
- Progress in the difficulty, to stimulate the challenge and maintain interest.
- Autonomy for students that make their own decisions, choosing the difficulty level and selecting their own learning pace.
Individual responsibility, since the students have the option of imposing their rate of personal work.

Error acceptance, as a natural and effective way of learning.

A.6 PLMan’s controller programming reference

This section details the programming interface designed for controllers to interact with PLMan and control Mr.PLMan. This programming interface is entirely designed in Prolog language and is only valid for controllers written in Prolog. These are the included modules and predicates:

- `- use_module('pl-man-game/main')`: this statement is required at the start of any controller knowledge base created in Prolog. It imports the Prolog programming interface for accessing information about Mr.PLMan’s environment and provides the predicates for selecting next actions.

- `see(S, D, O)`: this predicate is designed to provide the controller with information about its environment. $S \in \{ \text{normal, list} \}$ defines the kind of sensor to be used to obtain information. normal sensor explores information about the 9 adjacent cells to Mr.PLMan, whereas list sensor returns information about the 4 orthogonal lines of view Mr.PLMan has from its actual location. $D \in \{ \text{here, left, right, down, up, up-left, up-right, down-left, down-right} \}$ defines the exact adjacent cell or orthog-
ional line of view the controller wants to check. Finally, $O$ refers to the appearance of the object been seen at the desired cell, or to a list of objects in the case of the `list` sensor.

- `doAction(A)`: this predicate defines the interface the controller has to select the next action to perform during a given turn. Only one action is permitted per turn. Therefore, if the controller launches `doAction` more than once, only the last call is considered. $A$ is the action to be performed and has to be one of the following predicates:
  
  - `move(D)`: moves Mr.PLMan one cell in the given direction expressed by $D \in \{ \text{up, down, left, right, none} \}$. If the movement results in a collision with a solid object, Mr.PLMan does not move and a warning is issued with its correspondent punishment.
  
  - `get(D)`: makes Mr.PLMan pick the object located at the cell expressed by $D \in \{ \text{up, down, left, right, here} \}$. If Mr.PLMan already has an object in its inventory or there is no object in the cell expressed by $D$, a warning is issued along with its correspondent punishment.
  
  - `drop(D)`: makes Mr.PLMan drop the object it is presently carrying in the cell referenced by $D \in \{ \text{up, down, left, right, here} \}$. If Mr.PLMan is carrying no object in its inventory, a warning is issued and a punishment is added to the score.
  
  - `use(D)`: makes Mr.PLMan use the object it is presently carrying in the direction expressed by $D \in \{ \text{up, down, left, right} \}$. 


A.6. PLMan’s controller programming reference

If Mr. PLMan is carrying no object or is not possible to use the object in the desired direction, a warning is issued along with its correspondent punishment.

• **havingObject**(appearance(0)): this predicate succeeds when Mr. PLMan is carrying an object whose representative character is O (its appearance). This is useful to differentiate logic depending on the object Mr. PLMan is carrying. It has two variants that can also be used:
  
  – **havingObject**(name(ON)): its completely equivalent, but the check is done using the name of the object (a Prolog atom), instead of its representative character.
  
  – **havingObject**: this version of the predicate succeeds when Mr. PLMan is carrying an object, no matter which object is. This is useful for constructing logic when Mr. PLMan has no object also.
Appendix A. The PLMan Learning System
Appendix B

NEAT and HyperNEAT parameters

This appendix describes the free parameters that NEAT and HyperNEAT algorithms expect the researcher to configure at design time. These parameters control the behaviour of both algorithms and have a great impact on performance and learning capabilities for a given problem. They usually require be manually fine tuned before getting good results on any new problem faced.

B.1 NEAT Parameters

NEAT has the following free parameters that have to be configured by the researcher at design time:
• **Population Size**: total number of individuals that the genetic algorithm will manage as population. Problems requiring high variety of genes will require higher populations sizes.

• **Number of elite individuals**: sets the number of individual that are copied from one population to the next one during the generation of a new epoch. The chosen individuals are those with higher fitness. Setting this parameter to 0 will disable elitism.

• **Speciation** (0/1): enables or disables the generation of species with respect to compatibility thresholds between genomes. Species protect genetic innovations by giving a bonus to its individuals during early epochs of evolution.

• **Crossover Rate** ([0, 1]): this value is the probability a pair of individuals has of generating offspring through the crossover operator. This is evaluated each epoch, when producing population for the next epoch, after selecting each new pair of parents.

• **Add New Loop Rate** ([0, 1]): probability of adding a new loop link. This probability is evaluated for each neuron of each new genome produced when generated next population for a given epoch.

• **Add New Link Rate** ([0, 1]): probability of adding a new link between two neurons of a genome. This probability is evaluated for each new genome generated as offspring. When the genome is generated, this probability is evaluated and, in case it results positive, two random neurons are selected from the genome and a link connecting them is added.
• **Add New Neuron Rate** ([0, 1]): this probability works completely similar to the probability of adding a new link. However, when it results positive, it selects one random link from the genome and adds the new neuron in the middle of the link.

• **Weight Mutation Rate** ([0, 1]): probability of a concrete weight of the genome to be mutated. This probability is calculated for all the weights of each new genome generated as offspring.

• **Weight Substitution Probability** ([0, 1]): On the event of a weight being mutated, it may be substituted or just altered a little bit. This probability selects which one of these two cases will occur.

• **Weight Range**: This parameter controls the real-valued range in which newly generated weights must lay. It affects also newly generated weights on a weight substitution event.

• **Survival Rate** ([0, 1]): Proportion of individuals inside the same species that will be allowed to produce offspring for the next generation.

• **Maximum Stagnation**: Number of total generations that a species is permitted to continue evolving without producing better individuals. After these generations have passed, the species is deleted.

## B.2 HyperNEAT Parameters

HyperNEAT has more parameters than NEAT because it relies on a modification of NEAT that evolves Compositional Pattern Producing Net-
works (CPPNs). CPPNs are networks that produce patterns, and these patterns are used to generate the connectivity of greater neural networks, that are the ones finally produced as output by HyperNEAT. Therefore, NEAT parameters are required, as well as CPPN parameters, along with parameters to control the size and architecture of the greater neural networks produced. These are the specific parameters for HyperNEAT, not taking into account NEAT required parameters:

- **Expressions Threshold** ([0, 1]): values generated by a CPPN are considered to be weights of links in the produced neural network. These weights are only considered valid (not-0) if their value is greater than this threshold.

- **Function set**: set of mathematical functions that will be available to be used as nodes of the CPPN. This set is typically composed of continuous functions like Sine, Gaussian, Sigmoid or Absolute Value.

- **Substrates**: although this is not a formal parameter, it requires to be set at design time. A substrate is a 2D set of nodes to be used as a layer in the final neural network produced by HyperNEAT. If only one substrate is used, the final neural network will be a normal 2D neural network, like the ones that may be produced by NEAT or any other algorithm. If several substrates are used, the structure of the neural network becomes 3D, expanding structural possibilities of neural networks produced by other methods.
Appendix C

Detailed results

This appendix shows additional result graphs for representative mazes, as well as the complete results table with all information and similarity values. All these data is reported for further reference.

C.1 Complete results

Table C.1 sums up the complete experimental results obtained for this work. It includes results for the 120 mazes tested using NEAT and HyperNEAT. The fields included in the table are explained as follows:

- $z$: number of maze (or maze identifier)
- $\text{Stg}$: stage (1-2) in which the maze is classified by professors
- $\text{Dif}$: manually estimated difficulty value by professors (0-5)
- $\text{Nst}$: total number of students that have been assigned this maze
and have created at least 1 controller. Students that have been assigned the map but have never attempted to create a controller for it are not included.

- $t$: time of the last improvement created by a student. Each time a student creates a controller, it is evaluated. If the controller achieves a higher score than previous best score achieved, this value is updated. The aim is to count all the development time that students invested for creating their controllers and solving the maze.

- $T$: average time that students took for developing a successful controller for the maze. In this context, a successful controller is one which achieves a score over 75%.

- $S_{\text{min}}$: minimum score achieved by students. Is the minimum value of the scores achieved by all the students that submitted at least one controller to solve the maze. Therefore, this value is calculated as the minimum of the best scores achieved by each student. There is no $S_{\text{max}}$ because for each maze among the 120 studied, there is always one student at least that achieved 100% score.

- $\sigma_{\Theta(t)}^{\text{NEAT}}$: similarity calculated between measured difficulty for students and measured difficulty for NEAT. Function $\Theta$ is the transformation from $t$ (hours) to $t'$ (NEAT training epochs). A ‘–’ value means that NEAT was unable to score a single point in the maze, which invalidates the similarity measure.

- $\sigma_{\Theta'(t)}^{\text{Hyper}}$: similarity calculated between measured difficulty for students and measured difficulty for HyperNEAT. Function $\Theta'$ is the
C.1. Complete results  

transformation from $t$ (hours) to $t''$ (HyperNEAT training epochs). A ‘−’ value means that HyperNEAT was unable to score a single point in the maze, which invalidates the similarity measure.
### Table C.1: Complete results for mazes tested in this research: mazes 1-00 to 1-29

<table>
<thead>
<tr>
<th>$z$</th>
<th>Stg</th>
<th>Dif</th>
<th>Nst</th>
<th>$t$</th>
<th>$\bar{t}$</th>
<th>$S_{\text{min}}$</th>
<th>$\sigma_{\text{NEAT}}(\Theta(t))$</th>
<th>$\sigma_{\text{Hyper}}(\Theta(t))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-00</td>
<td>1</td>
<td>1</td>
<td>20</td>
<td>0.219</td>
<td>0.082</td>
<td>100</td>
<td>0.821</td>
<td>0.966</td>
</tr>
<tr>
<td>1-01</td>
<td>1</td>
<td>3</td>
<td>32</td>
<td>1.405</td>
<td>0.453</td>
<td>100</td>
<td>0.877</td>
<td>0.921</td>
</tr>
<tr>
<td>1-02</td>
<td>1</td>
<td>1</td>
<td>22</td>
<td>0.261</td>
<td>0.092</td>
<td>100</td>
<td>0.900</td>
<td>0.897</td>
</tr>
<tr>
<td>1-03</td>
<td>1</td>
<td>2</td>
<td>26</td>
<td>0.907</td>
<td>0.431</td>
<td>100</td>
<td>0.868</td>
<td>0.902</td>
</tr>
<tr>
<td>1-04</td>
<td>1</td>
<td>2</td>
<td>28</td>
<td>1.278</td>
<td>0.374</td>
<td>100</td>
<td>0.892</td>
<td>0.881</td>
</tr>
<tr>
<td>1-05</td>
<td>1</td>
<td>3</td>
<td>38</td>
<td>4.471</td>
<td>0.503</td>
<td>100</td>
<td>0.878</td>
<td>0.850</td>
</tr>
<tr>
<td>1-06</td>
<td>1</td>
<td>2</td>
<td>28</td>
<td>1.385</td>
<td>0.409</td>
<td>100</td>
<td>0.905</td>
<td>0.899</td>
</tr>
<tr>
<td>1-07</td>
<td>1</td>
<td>3</td>
<td>30</td>
<td>1.75</td>
<td>0.557</td>
<td>100</td>
<td>0.893</td>
<td>0.969</td>
</tr>
<tr>
<td>1-08</td>
<td>1</td>
<td>2</td>
<td>24</td>
<td>1.729</td>
<td>0.746</td>
<td>74</td>
<td>0.844</td>
<td>0.805</td>
</tr>
<tr>
<td>1-09</td>
<td>1</td>
<td>1</td>
<td>14</td>
<td>0.576</td>
<td>0.292</td>
<td>85</td>
<td>0.910</td>
<td>0.927</td>
</tr>
<tr>
<td>1-10</td>
<td>1</td>
<td>3</td>
<td>28</td>
<td>5.52</td>
<td>0.664</td>
<td>100</td>
<td>0.902</td>
<td>0.840</td>
</tr>
<tr>
<td>1-11</td>
<td>1</td>
<td>2</td>
<td>22</td>
<td>4.153</td>
<td>0.638</td>
<td>100</td>
<td>0.862</td>
<td>0.866</td>
</tr>
<tr>
<td>1-12</td>
<td>1</td>
<td>3</td>
<td>32</td>
<td>4.35</td>
<td>0.805</td>
<td>100</td>
<td>0.880</td>
<td>0.942</td>
</tr>
<tr>
<td>1-13</td>
<td>1</td>
<td>2</td>
<td>24</td>
<td>3.472</td>
<td>0.563</td>
<td>100</td>
<td>0.833</td>
<td>0.861</td>
</tr>
<tr>
<td>1-14</td>
<td>1</td>
<td>3</td>
<td>30</td>
<td>2.566</td>
<td>0.591</td>
<td>100</td>
<td>0.944</td>
<td>0.954</td>
</tr>
<tr>
<td>1-15</td>
<td>1</td>
<td>2</td>
<td>22</td>
<td>4.275</td>
<td>0.768</td>
<td>100</td>
<td>0.915</td>
<td>0.944</td>
</tr>
<tr>
<td>1-16</td>
<td>1</td>
<td>2</td>
<td>30</td>
<td>1.951</td>
<td>0.427</td>
<td>100</td>
<td>0.855</td>
<td>0.902</td>
</tr>
<tr>
<td>1-17</td>
<td>1</td>
<td>3</td>
<td>34</td>
<td>4.436</td>
<td>0.821</td>
<td>83</td>
<td>0.921</td>
<td>0.978</td>
</tr>
<tr>
<td>1-18</td>
<td>1</td>
<td>5</td>
<td>32</td>
<td>2.353</td>
<td>0.414</td>
<td>79</td>
<td>0.867</td>
<td>0.865</td>
</tr>
<tr>
<td>1-19</td>
<td>1</td>
<td>2</td>
<td>28</td>
<td>0.916</td>
<td>0.257</td>
<td>100</td>
<td>0.926</td>
<td>0.972</td>
</tr>
<tr>
<td>1-20</td>
<td>1</td>
<td>4</td>
<td>38</td>
<td>5.537</td>
<td>1.781</td>
<td>81</td>
<td>0.791</td>
<td>0.912</td>
</tr>
<tr>
<td>1-21</td>
<td>1</td>
<td>3</td>
<td>36</td>
<td>4.402</td>
<td>0.523</td>
<td>100</td>
<td>0.868</td>
<td>0.948</td>
</tr>
<tr>
<td>1-22</td>
<td>1</td>
<td>5</td>
<td>34</td>
<td>3.625</td>
<td>0.501</td>
<td>88</td>
<td>0.854</td>
<td>0.894</td>
</tr>
<tr>
<td>1-23</td>
<td>1</td>
<td>2</td>
<td>24</td>
<td>2.713</td>
<td>0.376</td>
<td>100</td>
<td>0.963</td>
<td>0.931</td>
</tr>
<tr>
<td>1-24</td>
<td>1</td>
<td>2</td>
<td>18</td>
<td>1.182</td>
<td>0.408</td>
<td>100</td>
<td>0.942</td>
<td>0.911</td>
</tr>
<tr>
<td>1-25</td>
<td>1</td>
<td>3</td>
<td>32</td>
<td>1.842</td>
<td>0.473</td>
<td>100</td>
<td>0.922</td>
<td>0.923</td>
</tr>
<tr>
<td>1-26</td>
<td>1</td>
<td>3</td>
<td>38</td>
<td>5.589</td>
<td>0.654</td>
<td>82</td>
<td>0.872</td>
<td>0.911</td>
</tr>
<tr>
<td>1-27</td>
<td>1</td>
<td>4</td>
<td>34</td>
<td>3.645</td>
<td>0.469</td>
<td>100</td>
<td>0.916</td>
<td>0.939</td>
</tr>
<tr>
<td>1-28</td>
<td>1</td>
<td>4</td>
<td>42</td>
<td>4.816</td>
<td>0.408</td>
<td>90</td>
<td>0.899</td>
<td>0.896</td>
</tr>
<tr>
<td>1-29</td>
<td>1</td>
<td>4</td>
<td>36</td>
<td>2.98</td>
<td>0.51</td>
<td>89</td>
<td>0.966</td>
<td>0.943</td>
</tr>
</tbody>
</table>
### Table C.2: Complete results for mazes tested in this research: mazes 1-30 to 1-59

<table>
<thead>
<tr>
<th>m</th>
<th>Stg</th>
<th>Dif</th>
<th>Nst</th>
<th>$t$</th>
<th>$\bar{t}$</th>
<th>$S_{\min}$</th>
<th>$\Theta_\Theta(t)$</th>
<th>$\Theta_\Theta'(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-30</td>
<td>1</td>
<td>2</td>
<td>20</td>
<td>0.967</td>
<td>0.369</td>
<td>100</td>
<td>0.875</td>
<td>0.936</td>
</tr>
<tr>
<td>1-31</td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>1.601</td>
<td>0.256</td>
<td>100</td>
<td>0.790</td>
<td>0.950</td>
</tr>
<tr>
<td>1-32</td>
<td>1</td>
<td>1</td>
<td>14</td>
<td>0.55</td>
<td>0.231</td>
<td>100</td>
<td>0.884</td>
<td>0.900</td>
</tr>
<tr>
<td>1-33</td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>2.008</td>
<td>0.312</td>
<td>100</td>
<td>0.800</td>
<td>0.886</td>
</tr>
<tr>
<td>1-34</td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>0.907</td>
<td>0.301</td>
<td>100</td>
<td>0.835</td>
<td>0.897</td>
</tr>
<tr>
<td>1-35</td>
<td>1</td>
<td>1</td>
<td>18</td>
<td>0.85</td>
<td>0.263</td>
<td>100</td>
<td>0.938</td>
<td>0.906</td>
</tr>
<tr>
<td>1-36</td>
<td>1</td>
<td>1</td>
<td>12</td>
<td>1.392</td>
<td>0.506</td>
<td>100</td>
<td>0.845</td>
<td>0.906</td>
</tr>
<tr>
<td>1-37</td>
<td>1</td>
<td>2</td>
<td>22</td>
<td>4.067</td>
<td>0.92</td>
<td>78</td>
<td>0.965</td>
<td>0.953</td>
</tr>
<tr>
<td>1-38</td>
<td>1</td>
<td>5</td>
<td>38</td>
<td>5.572</td>
<td>0.595</td>
<td>84</td>
<td>0.762</td>
<td>0.843</td>
</tr>
<tr>
<td>1-39</td>
<td>1</td>
<td>1</td>
<td>18</td>
<td>1.076</td>
<td>0.271</td>
<td>100</td>
<td>0.988</td>
<td>0.952</td>
</tr>
<tr>
<td>1-40</td>
<td>1</td>
<td>3</td>
<td>34</td>
<td>3.025</td>
<td>0.599</td>
<td>80</td>
<td>0.912</td>
<td>0.927</td>
</tr>
<tr>
<td>1-41</td>
<td>1</td>
<td>3</td>
<td>36</td>
<td>2.816</td>
<td>0.737</td>
<td>100</td>
<td>0.874</td>
<td>0.972</td>
</tr>
<tr>
<td>1-42</td>
<td>1</td>
<td>2</td>
<td>26</td>
<td>2.516</td>
<td>0.458</td>
<td>100</td>
<td>0.856</td>
<td>0.966</td>
</tr>
<tr>
<td>1-43</td>
<td>1</td>
<td>3</td>
<td>28</td>
<td>6.728</td>
<td>0.758</td>
<td>67</td>
<td>0.927</td>
<td>0.925</td>
</tr>
<tr>
<td>1-44</td>
<td>1</td>
<td>3</td>
<td>30</td>
<td>4.228</td>
<td>0.715</td>
<td>77</td>
<td>0.905</td>
<td>0.892</td>
</tr>
<tr>
<td>1-45</td>
<td>1</td>
<td>3</td>
<td>30</td>
<td>2.538</td>
<td>0.438</td>
<td>87</td>
<td>0.959</td>
<td>0.946</td>
</tr>
<tr>
<td>1-46</td>
<td>1</td>
<td>4</td>
<td>36</td>
<td>1.456</td>
<td>0.307</td>
<td>100</td>
<td>0.907</td>
<td>0.947</td>
</tr>
<tr>
<td>1-47</td>
<td>1</td>
<td>5</td>
<td>38</td>
<td>3.318</td>
<td>0.781</td>
<td>93</td>
<td>0.857</td>
<td>0.919</td>
</tr>
<tr>
<td>1-48</td>
<td>1</td>
<td>3</td>
<td>30</td>
<td>6.73</td>
<td>1.074</td>
<td>78</td>
<td>0.764</td>
<td>0.887</td>
</tr>
<tr>
<td>1-49</td>
<td>1</td>
<td>4</td>
<td>38</td>
<td>5.431</td>
<td>0.795</td>
<td>85</td>
<td>0.698</td>
<td>0.838</td>
</tr>
<tr>
<td>1-50</td>
<td>1</td>
<td>4</td>
<td>34</td>
<td>6.182</td>
<td>0.801</td>
<td>83</td>
<td>0.811</td>
<td>0.879</td>
</tr>
<tr>
<td>1-51</td>
<td>1</td>
<td>5</td>
<td>40</td>
<td>5.851</td>
<td>1.147</td>
<td>83</td>
<td>0.986</td>
<td>0.910</td>
</tr>
<tr>
<td>1-52</td>
<td>1</td>
<td>4</td>
<td>36</td>
<td>6.458</td>
<td>1.458</td>
<td>89</td>
<td>0.908</td>
<td>0.929</td>
</tr>
<tr>
<td>1-53</td>
<td>1</td>
<td>5</td>
<td>44</td>
<td>6.341</td>
<td>1.089</td>
<td>93</td>
<td>0.895</td>
<td>0.897</td>
</tr>
<tr>
<td>1-54</td>
<td>1</td>
<td>4</td>
<td>32</td>
<td>6.696</td>
<td>1.331</td>
<td>84</td>
<td>0.893</td>
<td>0.930</td>
</tr>
<tr>
<td>1-55</td>
<td>1</td>
<td>4</td>
<td>36</td>
<td>5.287</td>
<td>1.619</td>
<td>89</td>
<td>0.892</td>
<td>0.947</td>
</tr>
<tr>
<td>1-56</td>
<td>1</td>
<td>4</td>
<td>36</td>
<td>3.381</td>
<td>0.49</td>
<td>77</td>
<td>0.842</td>
<td>0.890</td>
</tr>
<tr>
<td>1-57</td>
<td>1</td>
<td>4</td>
<td>32</td>
<td>2.037</td>
<td>0.328</td>
<td>83</td>
<td>0.876</td>
<td>0.906</td>
</tr>
<tr>
<td>1-58</td>
<td>1</td>
<td>5</td>
<td>38</td>
<td>7.48</td>
<td>1.784</td>
<td>78</td>
<td>0.856</td>
<td>0.905</td>
</tr>
<tr>
<td>1-59</td>
<td>1</td>
<td>5</td>
<td>34</td>
<td>5.631</td>
<td>1.515</td>
<td>93</td>
<td>0.859</td>
<td>0.913</td>
</tr>
<tr>
<td>$z$</td>
<td>Stg</td>
<td>Dif</td>
<td>Nst</td>
<td>$t$</td>
<td>$\bar{t}$</td>
<td>$S_{min}$</td>
<td>$\sigma_{\Theta(t)}^{NEAT}$</td>
<td>$\sigma_{\Theta'(t)}^{Hyper}$</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>----------</td>
<td>----------</td>
<td>---------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>2-00</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0.767</td>
<td>0.498</td>
<td>100</td>
<td>0.844</td>
<td>0.954</td>
</tr>
<tr>
<td>2-01</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1.393</td>
<td>0.612</td>
<td>82</td>
<td>0.830</td>
<td>0.892</td>
</tr>
<tr>
<td>2-02</td>
<td>2</td>
<td>2</td>
<td>16</td>
<td>2.176</td>
<td>0.332</td>
<td>43</td>
<td>0.848</td>
<td>0.885</td>
</tr>
<tr>
<td>2-03</td>
<td>2</td>
<td>4</td>
<td>24</td>
<td>2.497</td>
<td>0.711</td>
<td>85</td>
<td>0.833</td>
<td>0.903</td>
</tr>
<tr>
<td>2-04</td>
<td>2</td>
<td>3</td>
<td>22</td>
<td>6.826</td>
<td>2.595</td>
<td>90</td>
<td>0.802</td>
<td>0.785</td>
</tr>
<tr>
<td>2-05</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>4.51</td>
<td>2.254</td>
<td>100</td>
<td>0.946</td>
<td>0.919</td>
</tr>
<tr>
<td>2-06</td>
<td>2</td>
<td>2</td>
<td>16</td>
<td>8.226</td>
<td>1.551</td>
<td>0</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>2-07</td>
<td>2</td>
<td>4</td>
<td>28</td>
<td>7.737</td>
<td>2.044</td>
<td>82</td>
<td>0.856</td>
<td>0.902</td>
</tr>
<tr>
<td>2-08</td>
<td>2</td>
<td>3</td>
<td>36</td>
<td>11.004</td>
<td>2.604</td>
<td>87</td>
<td>0.699</td>
<td>0.743</td>
</tr>
<tr>
<td>2-09</td>
<td>2</td>
<td>3</td>
<td>26</td>
<td>6.378</td>
<td>1.173</td>
<td>83</td>
<td>0.844</td>
<td>0.907</td>
</tr>
<tr>
<td>2-10</td>
<td>2</td>
<td>3</td>
<td>26</td>
<td>8.961</td>
<td>2.216</td>
<td>77</td>
<td>0.665</td>
<td>0.766</td>
</tr>
<tr>
<td>2-11</td>
<td>2</td>
<td>5</td>
<td>46</td>
<td>16.505</td>
<td>3.882</td>
<td>65</td>
<td>0.737</td>
<td>0.865</td>
</tr>
<tr>
<td>2-12</td>
<td>2</td>
<td>3</td>
<td>18</td>
<td>1.72</td>
<td>0.625</td>
<td>68</td>
<td>0.873</td>
<td>0.951</td>
</tr>
<tr>
<td>2-13</td>
<td>2</td>
<td>3</td>
<td>22</td>
<td>4.074</td>
<td>0.961</td>
<td>100</td>
<td>0.786</td>
<td>0.857</td>
</tr>
<tr>
<td>2-14</td>
<td>2</td>
<td>4</td>
<td>24</td>
<td>5.614</td>
<td>1.684</td>
<td>79</td>
<td>0.664</td>
<td>0.786</td>
</tr>
<tr>
<td>2-15</td>
<td>2</td>
<td>4</td>
<td>34</td>
<td>12.423</td>
<td>2.272</td>
<td>100</td>
<td>0.852</td>
<td>0.859</td>
</tr>
<tr>
<td>2-16</td>
<td>2</td>
<td>3</td>
<td>18</td>
<td>3.418</td>
<td>0.534</td>
<td>100</td>
<td>0.841</td>
<td>0.939</td>
</tr>
<tr>
<td>2-17</td>
<td>2</td>
<td>4</td>
<td>20</td>
<td>4.184</td>
<td>0.745</td>
<td>94</td>
<td>0.815</td>
<td>0.832</td>
</tr>
<tr>
<td>2-18</td>
<td>2</td>
<td>3</td>
<td>20</td>
<td>2.935</td>
<td>0.662</td>
<td>85</td>
<td>0.872</td>
<td>0.977</td>
</tr>
<tr>
<td>2-19</td>
<td>2</td>
<td>5</td>
<td>40</td>
<td>11.499</td>
<td>2.59</td>
<td>91</td>
<td>0.821</td>
<td>0.862</td>
</tr>
<tr>
<td>2-20</td>
<td>2</td>
<td>3</td>
<td>12</td>
<td>4.132</td>
<td>1.604</td>
<td>100</td>
<td>0.820</td>
<td>0.919</td>
</tr>
<tr>
<td>2-21</td>
<td>2</td>
<td>4</td>
<td>26</td>
<td>6.902</td>
<td>2.512</td>
<td>80</td>
<td>0.839</td>
<td>0.846</td>
</tr>
<tr>
<td>2-22</td>
<td>2</td>
<td>5</td>
<td>34</td>
<td>13.883</td>
<td>2.81</td>
<td>87</td>
<td>0.633</td>
<td>0.719</td>
</tr>
<tr>
<td>2-23</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>1.141</td>
<td>0.411</td>
<td>100</td>
<td>0.919</td>
<td>0.988</td>
</tr>
<tr>
<td>2-24</td>
<td>2</td>
<td>3</td>
<td>20</td>
<td>3.384</td>
<td>0.893</td>
<td>93</td>
<td>0.872</td>
<td>0.942</td>
</tr>
<tr>
<td>2-25</td>
<td>2</td>
<td>5</td>
<td>40</td>
<td>4.821</td>
<td>1.421</td>
<td>80</td>
<td>0.853</td>
<td>0.895</td>
</tr>
<tr>
<td>2-26</td>
<td>2</td>
<td>5</td>
<td>34</td>
<td>5.791</td>
<td>1.804</td>
<td>83</td>
<td>0.818</td>
<td>0.899</td>
</tr>
<tr>
<td>2-27</td>
<td>2</td>
<td>2</td>
<td>14</td>
<td>2.842</td>
<td>0.779</td>
<td>52</td>
<td>0.893</td>
<td>0.914</td>
</tr>
<tr>
<td>2-28</td>
<td>2</td>
<td>2</td>
<td>14</td>
<td>2.918</td>
<td>1.339</td>
<td>63</td>
<td>0.887</td>
<td>0.908</td>
</tr>
<tr>
<td>2-29</td>
<td>2</td>
<td>2</td>
<td>20</td>
<td>3.072</td>
<td>0.66</td>
<td>98</td>
<td>0.940</td>
<td>0.929</td>
</tr>
</tbody>
</table>

Table C.3: Complete results for mazes tested in this research: mazes 2-00 to 2-29
### Complete results

<table>
<thead>
<tr>
<th>z</th>
<th>Stg</th>
<th>Dif</th>
<th>Nst</th>
<th>t</th>
<th>(\bar{t})</th>
<th>(S_{\text{min}})</th>
<th>(\sigma_{\Theta(t)}^{\text{NEAT}})</th>
<th>(\sigma_{\Theta'(t)}^{\text{Hyper}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-30</td>
<td>2</td>
<td>3</td>
<td>28</td>
<td>4.974</td>
<td>0.73</td>
<td>82</td>
<td>0.921</td>
<td>0.920</td>
</tr>
<tr>
<td>2-31</td>
<td>2</td>
<td>2</td>
<td>12</td>
<td>3.353</td>
<td>0.971</td>
<td>100</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>2-32</td>
<td>2</td>
<td>3</td>
<td>28</td>
<td>6.554</td>
<td>1.643</td>
<td>77</td>
<td>0.971</td>
<td>0.972</td>
</tr>
<tr>
<td>2-33</td>
<td>2</td>
<td>3</td>
<td>22</td>
<td>5.755</td>
<td>1.815</td>
<td>92</td>
<td>0.900</td>
<td>0.946</td>
</tr>
<tr>
<td>2-34</td>
<td>2</td>
<td>3</td>
<td>12</td>
<td>4.219</td>
<td>1.236</td>
<td>84</td>
<td>0.908</td>
<td>0.916</td>
</tr>
<tr>
<td>2-35</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2.409</td>
<td>1.239</td>
<td>100</td>
<td>0.918</td>
<td>0.906</td>
</tr>
<tr>
<td>2-36</td>
<td>2</td>
<td>2</td>
<td>20</td>
<td>5.376</td>
<td>0.991</td>
<td>98</td>
<td>0.844</td>
<td>0.875</td>
</tr>
<tr>
<td>2-37</td>
<td>2</td>
<td>2</td>
<td>14</td>
<td>3.888</td>
<td>1.086</td>
<td>91</td>
<td>0.839</td>
<td>0.880</td>
</tr>
<tr>
<td>2-38</td>
<td>2</td>
<td>2</td>
<td>24</td>
<td>4.11</td>
<td>1.017</td>
<td>93</td>
<td>0.899</td>
<td>0.955</td>
</tr>
<tr>
<td>2-39</td>
<td>2</td>
<td>2</td>
<td>18</td>
<td>4.713</td>
<td>0.905</td>
<td>80</td>
<td>0.883</td>
<td>0.920</td>
</tr>
<tr>
<td>2-40</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1.125</td>
<td>0.666</td>
<td>100</td>
<td>0.804</td>
<td>0.942</td>
</tr>
<tr>
<td>2-41</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>7.14</td>
<td>1.396</td>
<td>91</td>
<td>0.850</td>
<td>0.918</td>
</tr>
<tr>
<td>2-42</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1.471</td>
<td>0.987</td>
<td>100</td>
<td>0.889</td>
<td>0.945</td>
</tr>
<tr>
<td>2-43</td>
<td>2</td>
<td>3</td>
<td>22</td>
<td>6.264</td>
<td>1.266</td>
<td>86</td>
<td>0.768</td>
<td>0.831</td>
</tr>
<tr>
<td>2-44</td>
<td>2</td>
<td>5</td>
<td>40</td>
<td>10.013</td>
<td>2.16</td>
<td>99</td>
<td>0.883</td>
<td>0.884</td>
</tr>
<tr>
<td>2-45</td>
<td>2</td>
<td>1</td>
<td>14</td>
<td>3.234</td>
<td>1.035</td>
<td>85</td>
<td>0.963</td>
<td>0.958</td>
</tr>
<tr>
<td>2-46</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>1.577</td>
<td>0.537</td>
<td>0</td>
<td>0.910</td>
<td>0.910</td>
</tr>
<tr>
<td>2-47</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>2.092</td>
<td>0.769</td>
<td>86</td>
<td>0.889</td>
<td>0.900</td>
</tr>
<tr>
<td>2-48</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4.448</td>
<td>1.263</td>
<td>83</td>
<td>0.843</td>
<td>0.920</td>
</tr>
<tr>
<td>2-49</td>
<td>2</td>
<td>3</td>
<td>16</td>
<td>5.543</td>
<td>1.823</td>
<td>81</td>
<td>0.827</td>
<td>0.861</td>
</tr>
<tr>
<td>2-50</td>
<td>2</td>
<td>2</td>
<td>18</td>
<td>5.295</td>
<td>1.746</td>
<td>78</td>
<td>0.878</td>
<td>0.931</td>
</tr>
<tr>
<td>2-51</td>
<td>2</td>
<td>5</td>
<td>44</td>
<td>13.602</td>
<td>2.707</td>
<td>84</td>
<td>0.872</td>
<td>0.886</td>
</tr>
<tr>
<td>2-52</td>
<td>2</td>
<td>3</td>
<td>22</td>
<td>5.19</td>
<td>1.154</td>
<td>100</td>
<td>0.831</td>
<td>0.880</td>
</tr>
<tr>
<td>2-53</td>
<td>2</td>
<td>4</td>
<td>18</td>
<td>6.927</td>
<td>1.476</td>
<td>100</td>
<td>0.836</td>
<td>0.860</td>
</tr>
<tr>
<td>2-54</td>
<td>2</td>
<td>4</td>
<td>22</td>
<td>9.411</td>
<td>3.68</td>
<td>94</td>
<td>0.840</td>
<td>0.835</td>
</tr>
<tr>
<td>2-55</td>
<td>2</td>
<td>4</td>
<td>22</td>
<td>6.875</td>
<td>1.002</td>
<td>62</td>
<td>0.712</td>
<td>0.800</td>
</tr>
<tr>
<td>2-56</td>
<td>2</td>
<td>4</td>
<td>16</td>
<td>7.412</td>
<td>2.289</td>
<td>84</td>
<td>0.697</td>
<td>0.815</td>
</tr>
<tr>
<td>2-57</td>
<td>2</td>
<td>4</td>
<td>24</td>
<td>8.042</td>
<td>1.592</td>
<td>86</td>
<td>0.746</td>
<td>0.855</td>
</tr>
<tr>
<td>2-58</td>
<td>2</td>
<td>2</td>
<td>14</td>
<td>4.579</td>
<td>2.059</td>
<td>91</td>
<td>0.722</td>
<td>0.767</td>
</tr>
<tr>
<td>2-59</td>
<td>2</td>
<td>4</td>
<td>30</td>
<td>12.858</td>
<td>2.338</td>
<td>83</td>
<td>0.773</td>
<td>0.889</td>
</tr>
</tbody>
</table>

Table C.4: Complete results for mazes tested in this research: mazes 2-30 to 2-59
C.2 Additional experiments analysed

A total of 10 mazes have been selected and analysed with detailed graphs. Each maze has been selected as being representative for its stage and difficulty level. It is important to notice that stages and difficulty levels are classifications established manually by experts (professors in this case). Selecting a representative for each level has the potential to let us compare actual measured difficulties with expert-assigned values.

The following enumeration gives a brief interpretation of the results obtained for each of the mazes detailed graphically:

- Maze 1-31 (figure C.1): NEAT finds this maze easier to solve than HyperNEAT, but HyperNEAT better estimates students’ actual difficulty progression. In fact, HyperNEAT result is impressive ($\sigma = 0.95$) and establishes a highly accurate estimation for actual students’ difficulty.

- Maze 1-08 (figure C.2): Both algorithms seem to have problems to deal with the blocking enemy ($E$). However, when they manage dodge the enemy, their difficulty progression continues parallel to students’. In fact, measures suggest that students also have a minimum difficulty to dodge the enemy, but they manage to solve it way earlier. Although final similarity results are not high $\sigma = 0.844$, graphical results show highly valuable information.

- Maze 1-56 (figure C.3): NEAT seems to struggle more than HyperNEAT on finding the key ($a$) and managing to open the door ($-$). Also, starting progression is slower than students’, probably due
to the fact that students usually start by reusing controllers developed for previous mazes. However, students also show difficulty to find complete solutions: developing a controller that eats most of the dots is easy for students, but developing one that eats all the dots requires much effort. With respect to this, HyperNEAT again shows a great parallelism with students’ behaviour (and a nice estimation $\sigma = 0.89$).

- Maze 1-59 (figure C.4): It results very interesting to see different behaviours for NEAT and HyperNEAT in this maze. HyperNEAT finds a great solution faster than students whereas NEAT seems to have difficulties to dodge patrolling enemies at the start. However, both algorithms require time to find perfect solutions, same way as students. Being HyperNEAT nearer to students’ behaviour, NEAT ends up with almost the same $D_t$ value at the end. Similarity for this maze ($\sigma = 0.913$) shows HyperNEAT acting again as a nice estimator for actual students’ behaviour.

- Maze 2-40 (figure C.5): Interestingly, this maze seems easier for NEAT than it is for students, whereas HyperNEAT seems to struggle at the start. The symmetry of the maze and Mr.PLMan starting from the centre seems to confuse HyperNEAT that develops many suboptimal solutions that leave out isolated dot pools at the start. This leads to a completely different progress for both algorithms at the start, up to $t'' \approx 190$, when HyperNEAT finds a new type of controller that works pretty similar to NEAT’s. HyperNEAT gives again a very accurate similarity value $\sigma = 0.942$, despite behaving
in an apparently erratic way.

- Maze 2-48 (figure C.6): This maze has 3 corridors disposed horizontally that seem to trick NEAT. First controllers generated by NEAT eat all upper dots, forgetting the 2 lower corridors. Then, it has problems to find controllers that enter these 2 lower corridors and get the ball (o). NEAT ends up unable to launch the ball to goal (l) properly, failing to score and to pass to the chamber with the last 9 dots. Curiously, this is similar to students, because some of them did not achieve a 100% score either. HyperNEAT does not seem to have trouble with corridors, but requires plenty of epochs to learn how to use the ball and score a goal. This appears being not only costly for HyperNEAT but also for students, which finally gives a pretty similar result ($\sigma = 0.92$) in spite of following apparently different paths for solving the maze.

- Maze 2-49 (figure C.7): The curious thing about this maze is that students do not manage to have good controllers in a short period of time. It seems that they plan on taking the mine (+) first and struggle with the patrolling movement of the enemies (E). NEAT and HyperNEAT manage to get all the dots outside the 2 doors (| -) very fast, which is shown in the graphs as a difficulty decreasing faster than students’’. However, this behaviour has also a drawback for NEAT in this case. The algorithm becomes unable to correctly take and use the mine to break the walls that protect the key (a). HyperNEAT seems to find the same issue, but manages to find a solution after being stuck for many epochs. Increasing complexity
of the mazes seems to undermine the ability of the algorithms to act as reasonable estimators for students’ behaviour ($\sigma = 0.861$).

- Maze 2-56 (figure C.8): Surprisingly, both algorithms find it difficult to take and use the key (a) to open the first door (|). The disposition of this part of the maze may add up some movement complexity, because students also seem to have some dispute with it, though being it lighter. In this case, both algorithms end up unable to use the mine (+) to get rid of the follower phantoms (F) sited on the last part of the maze. However, HyperNEAT finds a way to eat the last inner dots by making phantoms follow Mr.PLMan, leaving the dots free for being eaten. This time, the behaviour of both algorithms is far from representing a good estimation for students’ behaviour ($\sigma = 0.815$). Again, more complexity seems to mean more difficulty to obtain a proper estimation.
Figure C.1: Difficulty and similarity results for NEAT and HyperNEAT on maze 1-31 (Stage 1, difficulty 1).
C.2. Additional experiments analysed

Figure C.2: Difficulty and similarity results for NEAT and HyperNEAT on maze 1-08 (Stage 1, difficulty 2).
Figure C.3: Difficulty and similarity results for NEAT and HyperNEAT on maze 1-56 (Stage 1, difficulty 4).
C.2. Additional experiments analysed

Figure C.4: Difficulty and similarity results for NEAT and HyperNEAT on maze 1-59 (Stage 1, difficulty 5).
Figure C.5: Difficulty and similarity results for NEAT and HyperNEAT on maze 2-40 (Stage 2, difficulty 1).
C.2. Additional experiments analysed

Figure C.6: Difficulty and similarity results for NEAT and HyperNEAT on maze 2-48 (Stage 2, difficulty 2).
Figure C.7: Difficulty and similarity results for NEAT and HyperNEAT on maze 2-49 (Stage 2, difficulty 3).
Figure C.8: Difficulty and similarity results for NEAT and HyperNEAT on maze 2-56 (Stage 2, difficulty 4).
Appendix C. Detailed results
D.1 Resumen

En cualquier entorno de aprendizaje o formación, las actividades de formación son la base para el aprendizaje práctico. Los estudiantes necesitan practicar para adquirir nuevas habilidades y perfeccionar las adquiridas previamente. La clave para optimizar el proceso de aprendizaje consiste en realizar una correcta asignación de actividades de formación a los estudiantes. Cada estudiante tiene necesidades específicas en función de sus conocimientos previos y habilidades personales. Una asignación correcta para un estudiante concreto consistiría en seleccionar una actividad de formación que se ajusta a las habilidades y conocimientos del estudiante. Esto se refiere al concepto de dificultad. La dificultad de una actividad de formación se podría definir como el esfuerzo que un estudiante tiene que realizar para completar con éxito la actividad y obtener los resultados.
de aprendizaje asociados. Por lo tanto, una actividad difícil simplemente requiere mucho esfuerzo para ser completada con éxito.

Aquellos estudiantes que reciben actividades de formación demasiado difíciles tienden a abandonar en lugar de realizar el esfuerzo necesario. Esta situación se entiende mejor observando que el alumno percibe la actividad desequilibrada en cuanto a su relación esfuerzo-recomensa: demasiado esfuerzo para los resultados de aprendizaje esperados. Un caso similar se produce cuando la actividad es demasiado fácil. En ese caso, el esfuerzo estimado es bajo, pero los resultados de aprendizaje son aún menores. Si la actividad no representa un reto para el estudiante se debe a que éste ya domina las habilidades implicadas. Esto hace que los resultados del aprendizaje tienden a cero. Ambas situaciones conducen a los estudiantes a perder el interés.

Para evitar que esto suceda, los profesores y formadores estiman la dificultad de las actividades de formación basándose en su propia experiencia. Sin embargo, este procedimiento sufre un efecto llamado la Maldición del Conocimiento: cada persona que consigue dominar una actividad adquiere un sesgo a la hora de estimar el esfuerzo necesario para dominar esa misma actividad. Por lo tanto, estimar correctamente la dificultad de las actividades de formación es una tarea propensa a errores cuando se hace utilizando la propia experiencia. Por otra parte, estimar la dificultad sin haber realizado la actividad de formación probablemente produzca resultados aún peores.

Para escapar de este círculo vicioso, una primera solución sería medir el esfuerzo requerido para completar con éxito la actividad de formación.
Para ello debe definirse una medida de esfuerzo objetiva. Este enfoque en concreto ha sido seguido por muchos trabajos previos y uno de los enfoques principales en Learning Analytics. A pesar de que este enfoque ha producido muchos resultados de interés, tiene un serio inconveniente. Resulta imposible medir una actividad sin que algunos estudiantes la realicen. Así pues, en las etapas de diseño de la actividad, ¿Cómo sabe el diseñador si la actividad es demasiado difícil o demasiado fácil? ¿Hay alguna manera de tener una estimación válida de la dificultad de una actividad de formación antes de probarla con estudiantes?

Este trabajo propone un nuevo enfoque para abordar este problema. El enfoque consiste en entrenar un algoritmo de Machine Learning y medir el “esfuerzo” que el algoritmo necesita para entrenar hasta que consigue solucionar la actividad de formación propuesta. El “esfuerzo” será el coste de aprendizaje: el tiempo que el algoritmo requiere para entrenar y ajustarse. Después de eso, los resultados obtenidos por el algoritmo de Machine Learning deben compararse con los resultados medidos en estudiantes reales. Suponiendo que los costes de aprendizaje para el algoritmo de Machine Learning y para los estudiantes tuvieran algún tipo de correlación entre sí, los resultados de la comparación deberían demostrar la existencia de esa correlación. Si ese fuera el caso, entonces el coste de entrenamiento del algoritmo de Machine Learning podría ser utilizado como una estimación de la dificultad de la actividad de formación para los estudiantes.

Para poner en práctica este enfoque y obtener datos experimentales, dos algoritmos de Neuroevolución han sido seleccionados como algorit-
mos de Machine Learning: Neuroevolution of Augmenting Topologies (NEAT) y Hypercube-based Neuroevolution of Augmenting Topologies (HyperNEAT).

La implementación del enfoque propuesto ha dado lugar a muchas contribuciones que se presentan en este trabajo:

- Una nueva definición de la dificultad como función, basada en los progresos realizados en el tiempo como medida inversa del esfuerzo/coste de aprendizaje.

- Una medida de similitud para comparar los resultados de los algoritmos de Machine Learning y los de los estudiantes, para conocer la precisión de la estimación.

- Un juego llamado PLMan que se utiliza como actividad de formación para los experimentos. Es un juego estilo Pacman compuesto por hasta 220 laberintos diferentes. El juego se utiliza para enseñar programación Prolog, Lógica y una introducción a luz de la Inteligencia Artificial.

- Una aplicación de NEAT y HyperNEAT para aprender a resolver automáticamente laberintos del juego PLMan.

- Una nueva aplicación de la Neuroevolución para estimar la dificultad de las actividades de formación en etapas de diseño.

Los resultados experimentales presentados confirman que existe una correlación entre el costes de entrenamiento de los algoritmos de Neuroevolución y el coste de aprendizaje de los estudiantes. La bondad de
los resultados obtenidos está limitada por el alcance de este estudio y su naturaleza empírica. Sin embargo, se trata de resultados trascendentes, que pueden llegar a abrir una nueva línea de investigación sobre la relación entre el Machine Learning y los seres humanos con respecto al proceso de aprendizaje en sí mismo.

D.2 Introducción

El aprendizaje es un evento fascinante que sucede en la naturaleza como la máxima expresión de la adaptación. Tener la capacidad de aprender significa ser capaz de recordar eventos pasados y asociarlos con situaciones actuales para tomar decisiones. La mayoría de los animales muestran esta capacidad con diferentes grados de rendimiento. Intuitivamente, parece ser una característica clave para la subsistencia de cualquier criatura. Eso sería, según Darwin (1859), una buena razón para explicar la generalización de la Inteligencia en la mayoría de las criaturas en la Tierra.

El interés en cómo sucede el Aprendizaje ha estado presente en la cultura humana desde los primeros estudios conocidas. Aunque la investigación ha logrado muchos resultados interesantes relacionados con el aprendizaje en sí, sigue siendo mayoritariamente desconocida forma en que realmente sucede. Esto deja a la formación y la educación en la esfera de los conocimientos prácticos, todavía muy lejos de tener una comprensión científica de bajo nivel completa sobre cómo y por qué ocurre el aprendizaje. Por lo tanto, se requiere de mucha investigación en dos líneas principales: 1) estudios empíricos que reúnan más información ac-
erca de las conexiones indirectas entre la actividad y el aprendizaje, y 2) la investigación teórica para encontrar mejores explicaciones de alto nivel de este fenómeno.

Nuestra experiencia personal sugiere que existe una fuerte correlación entre las actividades que cualquier individuo realiza y el aprendizaje. Tiene sentido que el tener más experiencias proporcione más información y mejore la capacidad de recordar y relacionar recuerdos con situaciones futuras. Esta es la base para la formación y la educación: la creación de experiencias para que los cerebros de los alumnos se adapten y recuerden. De hecho, las experiencias de aprendizaje diseñadas por profesores/entrenadores con la intención de generar resultados de aprendizaje sobre quienes las realizan son a menudo llamadas actividades de formación.

El espacio de posibles experiencias de aprendizaje para crear es potencialmente infinito. ¿Cómo saber qué experiencias serán útiles para situaciones futuras? Suponiendo dos experiencias con el mismo resultado de aprendizaje, ¿cuál es mejor? ¿Por qué? ¿Hay una manera de buscar experiencias óptimas de aprendizaje? Estas preguntas guían la investigación académica sobre el aprendizaje. En estas preguntas, y asumiendo que todavía no hay ninguna explicación teórica sobre la manera exacta en que sucede aprendizaje, existe un cierto consenso que se resume en varias lecciones prácticas:

- Los profesionales asumen que el aprendizaje depende del estado previo de los cerebros de los estudiantes (es decir, su conocimiento previo) (Ley and Kump, 2013, Prensky, 2001, Redecker et al., 2012).
D.2. Introducción

Esto implica que las experiencias óptimas de aprendizaje, en el supuesto de que existan, serán diferentes para cada alumno.

- La motivación de los estudiantes parece jugar un papel clave en la eficacia de las experiencias de aprendizaje. La experiencia parece indicar que los estudiantes motivados a logran mejores resultados de aprendizaje de forma más rápida (Prensky, 2001).

- La motivación parece estar afectada por la sensación de progreso (Cocea and Weibelzahl, 2006, Koster and Wright, 2004, Wang and Newlin, 2000). Los estudiantes que sienten que su esfuerzo vale la pena en términos de progreso parecen aumentar su motivación y estar dispuestos a invertir un mayor esfuerzo en actividades de aprendizaje.

- En última instancia, la sensación de progreso se ve afectada por la naturaleza de las actividades de aprendizaje realizadas (Hu et al., 2014, Koster and Wright, 2004). Cuando son muy adecuadas para el estudiante, la motivación aparece, las actividades son realizadas y el alumno obtiene los resultados del aprendizaje.

- Una actividad de aprendizaje es considerada como muy adecuada para un estudiante cuando coincide con los conocimientos y capacidades previos del estudiante, y los conecta con un nuevo conocimiento o un nuevo nivel de desarrollo con respecto a alguna habilidad. Esto podría verse como un escalón, que hace coincidir los escalones siguientes y anteriores en una escalera, o como un eslabón, que engarza los enlaces anteriores y siguientes en una cadena.
Teniendo en cuenta el concepto de actividades de formación muy adecuadas para un estudiante, aparece la noción intuitiva de dificultad. La dificultad se podría definir como el coste \(^1\) que el estudiante tiene que pagar a fin de realizar con éxito la actividad de formación y adquirir sus resultados de aprendizaje. Esto también está conectado con los conocimientos previos requeridos: si una actividad requiere un conocimiento que el estudiante no posee, estos tendrán que ser adquiridos para poder realizar la actividad, lo que aumenta el coste.

Una actividad es considerada difícil cuando se requiere un gran coste para realizarla con éxito. A la inversa, una actividad sería fácil si requiere poco esfuerzo. Aunque estas consideraciones tienden a ser subjetivas y dependen de cada estudiante, este concepto es universalmente conocido y utilizado. Por otra parte, este concepto tiene implicaciones en todas las lecciones prácticas anteriormente enumeradas:

- Una actividad es demasiado difícil si requiere demasiado esfuerzo desde el punto de vista del estudiante. Esto significa que el estudiante puede percibir que la actividad ofrece una mala relación coste-beneficio. También podría significar que la actividad requiere que el esfuerzo se realice dentro de un plazo de tiempo muy limitado, lo que podría ser imposible o casi imposible para el estudiante.

- Por el contrario, una actividad demasiado fácil requeriría casi ningún esfuerzo por parte del estudiante. Esto generalmente significa que el estudiante ya posee los resultados de aprendizaje de la activi-

\(^1\)Coste aquí debe interpretarse como cantidad de esfuerzo: una combinación de tiempo dedicación y concentración requeridos.
dad produce. Aunque la actividad requiera una mínima cantidad de tiempo para ser realizada, podría ser considerada como mala en relación coste-beneficio: invertir un poco de tiempo para obtener casi ningún resultado de aprendizaje no parece una buena inversión.

Como consecuencia de esto, una actividad muy adecuada para un estudiante dado debe ser ni muy fácil ni muy difícil. Es ampliamente aceptado que las actividades con la dificultad apropiada tienen el potencial de motivar a los estudiantes. Por definición, estas actividades se ajustan con precisión las habilidades del estudiante, por lo que tienen una relación perfecta coste-beneficio. Esto puede ser una explicación válida sobre por qué estas actividades tienen tal potencial de motivación: son percibidas como una manera óptima para adquirir conocimientos o habilidades. Por otra parte, la retroalimentación que se produce al realizar estos tipos de actividades de formación también contribuye a producir una sensación positiva de progreso.

El problema surge cuando se intenta medir y/o estimar la dificultad para cualquier actividad de formación dada (Aponte et al., 2009, Missura and Gartner, 2011, Mladenov and Missura, 2010, Mourato and dos Santos, 2010, Ravi and S., 2013). Como se mencionó antes, el concepto universal de dificultad es totalmente subjetivo y, por lo general, se define por comparación entre las diferentes actividades de formación y los estudiantes que las realizan. En general, es fácil encontrar diferentes estudiantes para cualquier actividad de formación que indiquen valores diferentes cuando se le pregunte por la dificultad de una actividad realizada. Claramente, la medición y estimación de dificultad es una tarea
subjetiva: la forma habitual para estimar la dificultad de cualquier actividad de formación es por estimaciones basadas en la experiencia de quien las realiza. Profesores y entrenadores estiman la dificultad de las actividades de formación basándose en su propia experiencia en el tema. Sin embargo, utilizar la experiencia propia como fuente para medir la dificultad de cualquier actividad de aprendizaje es una tarea afectada por la Maldición del Conocimiento (Colin Camerer, 1989). Una vez que una persona domina una actividad de formación, la dificultad para conseguir dominarla desaparece. Por lo tanto, cuando la persona reflexiona sobre la dificultad de cualquier actividad de formación ya dominada, las estimaciones tenderán a estar polarizadas por el conocimiento adquirido (generalmente, aportando un sesgo de subestimación). No es sorprendente que estas estimaciones resulten ser inexactas para la mayoría de los estudiantes.

Una manera de abordar esto es mediante la medición a través de indicadores estadísticos realizados utilizando resultados históricos de los estudiantes (Cheng et al., 2008, Lykourentzou et al., 2009, Ravi and S., 2013, Romero et al., 2013). Basándose en los resultados se pueden obtener muchas mediciones interesantes: tiempo requerido para realizar una actividad, porcentajes de éxito, número de intentos fallidos antes de tener éxito ... Todas estas medidas dependen de la actividad en sí, pero la cuestión es conseguir suficientes datos objetivos para deducir indicadores comparativos entre actividades. Esto se ha hecho muchas veces con diferentes grados de éxito y utilidad (Cheng et al., 2008, Hu et al., 2014, Lykourentzou et al., 2009, Park and Kanehisa, 2003, Romero et al.,
Como sugieren los trabajos previos, este enfoque es muy prometedor, ya que es la base para un nuevo campo completo llamado Learning Analytics (Siemens, 2012).

El campo de Learning Analytics (Siemens, 2012) a menudo se centra en el análisis de estudiantes y actividades de formación a través de la aplicación de la estadística y el Machine Learning a los datos recogidos. Conseguir un corpus bien definido de los datos acerca de una actividad de formación permite una mejor comprensión de la actividad en sí, sus resultados de aprendizaje y sus dificultades. Este es un gran avance, pero con un coste: algunos estudiantes tienen que ser “sacrificados” para obtener los suficientes datos. Cuando se crea una nueva actividad, no hay evidencia empírica acerca de su dificultad y/o resultados de aprendizaje. Debido a esto, un valor inicial estimado de dificultad tiende a ser asignado manualmente por expertos sobre la base de su experiencia. Como se trata de un método propenso a errores, los primeros estudiantes que se enfrentan las nuevas actividades tienen que pagar el coste adicional de las estimaciones erróneas de dificultad.

Esto lleva a una pregunta fundamental: ¿hay otra manera de obtener estimaciones sobre la dificultad de las actividades de formación en las etapas de diseño? Si existe, ¿podría ser automatizado? ¿Podría ser optimizado y mejorado? Producir estimaciones de forma automática representaría una contribución interesante: liberaría tiempo de los profesores y formadores. El tiempo liberado pasaría a estar disponible para otras tareas más importantes como el diseño de nuevas actividades, por ejemplo. Por otra parte, la automatización de las estimaciones es sólo un
primer paso: una vez que están automatizadas, los investigadores pueden profundizar en el conocimiento sobre cómo funciona realmente el aprendizaje. Esto puede conducir a nuevos descubrimientos y crear un bucle de retroalimentación que podrían en última instancia transformar todo el sistema de enseñanza-aprendizaje.

Esta investigación comienza hipotetizando que hay maneras de definir la dificultad para poder ser medida y estimada de forma automática. En este contexto, estimar se refiere a la capacidad de predecir el valor final de dificultad antes de recoger datos reales \(^2\). Todo el trabajo llevado a cabo en esta investigación se hace sobre la base de que esto es posible, y el objetivo es conseguir una primera validación empírica de esta hipótesis. Validar que esto puede ser posible en la práctica, incluso si demuestra limitado a unos datos específicos, es un primer paso para motivar e invitar a realizar más investigación en este campo. Dirigir la atención a esta posibilidad inexplorada puede conducir a los investigadores producir más experiencias prácticas y resultados y, finalmente, al desarrollo de un corpus teórico de trabajo.

**D.2.1 Objetivos e hipótesis**

Comencemos asumiendo la hipótesis previamente enunciada de que existen definiciones de dificultad que nos permiten automatizar su medición y estimación. Entonces, el objetivo de este trabajo es el diseño y realización de experimentos para probar esta hipótesis empíricamente. La

\(^2\)Debería ser posible realizar predicciones para una actividad de formación dada durante las etapas de diseño.
validación se limitará al contexto propuesto, pero se considera suficiente para este trabajo, ya que es un punto de partida sólido para futuras investigaciones.

Para alcanzar el objetivo final de validar o descartar la hipótesis planteada, este será el plan a seguir:

- Crear una definición de dificultad para un actividad de formación que se pueda utilizar para medir el coste de aprendizaje de un estudiante genérico. Un estudiante genérico sería cualquier agente que realiza la actividad de formación y mejora sus resultados. Esta mejora es consecuencia del aprendizaje, sea realizado por un algoritmo de Machine Learning o por un humano.

- Definir indicadores generales y funciones para medir la dificultad, y derivar algunas implementaciones específicas para los problemas y conjuntos de datos propuestos. Para esta tarea será importante diseñar concretamente las propiedades deseadas que los indicadores y funciones deben poseer. El objetivo principal es la creación de procedimientos para medir y estimar de forma automática dificultad.

- Seleccionar un contexto particular y aplicar Machine Learning para estimar la dificultad de una actividad de formación específica. Producir adaptaciones de las funciones y los indicadores necesarios para la actividad seleccionada y para los algoritmos de Machine Learning que se utilizarán para generar las estimaciones.

- Diseñar indicadores para medir la precisión de las estimaciones real-
izadas utilizando Machine Learning en el paso anterior. Las estimaciones deben considerarse mejores a medida que sean más próximas a los datos medidos reales. Por lo tanto, los indicadores tendrán que medir la similitud entre las estimaciones y las medidas obtenidas a partir de datos reales.

- Por último, analizar las mediciones de similitud y buscar correlaciones entre las estimaciones y los datos recogidos reales. La existencia de correlaciones validaría la hipótesis en el contexto de los datos específicos utilizados en este trabajo.

Todos estos objetivos se basan en la hipótesis principal de este trabajo, que puede describirse del siguiente modo:

La etapa de entrenamiento de un algoritmo de Machine Learning se puede interpretar como el coste que el algoritmo tiene que pagar para aprender. Los seres humanos también tienen que pagar un coste para aprender, en forma de tiempo invertido y el esfuerzo realizado. Están estos dos costes relacionados en algún sentido? ¿Es posible encontrar correlaciones entre ellos? Si existen correlaciones, ¿son lo suficientemente fuertes como para estimar con precisión los costes de aprendizaje para los seres humanos a partir de los que tienen los programas?

Este trabajo asume que la respuesta a estas preguntas es sí, y tiene como objetivo demostrarlo empíricamente en un contexto práctico propuesto.
D.2.2 Contribución principal e impacto esperado

Este trabajo propone una nueva forma de definir la dificultad de las actividades de formación con el fin de poder medirlas y producir predicciones y estimaciones sobre su valor. Todo el trabajo se basa en la idea de que el Machine Learning y el aprendizaje humano están correlacionados de algún modo. Entendiendo la dificultad como un coste para alcanzar los resultados de aprendizaje, este trabajo propone que el coste que los algoritmos de Machine Learning tienen que pagar por aprender puede estar correlacionado con el coste que pagan los seres humanos.

La principal contribución de este trabajo es la demostración de que los costes propuestos están correlacionados, por lo menos para un conjunto de datos específico dado. Puesto que el conjunto de datos propuesto proviene de actividades de formación reales que se están llevando a cabo en la Universidad de Alicante, la existencia de una correlación debe ser considerada un evento inicialmente significativo. Esto es así debido a que un conjunto de datos de este tipo podría ser considerada como una muestra aleatoria de la distribución de todos los conjuntos de datos procedentes de posibles escenarios de aprendizaje reales. Si no hubiera correlación entre el Machine Learning y los seres humanos en sus costes de aprendizaje, habría una alta probabilidad para cualquier muestra de que no mostrar ninguna correlación. Por lo tanto, asumiendo que la muestra propuesta no tiene ningún sesgo específico, la existencia de una correlación es un primer paso importante hacia el uso de algoritmos de Machine Learning como estimadores de la dificultad de las actividades de formación en el momento de su creación.
Los resultados de esta investigación sugieren que existe una correlación entre los seres humanos y el Machine Learning respecto a los costes de aprendizaje. Puesto que esto habilita el uso de algoritmos de Machine Learning como estimadores de la dificultad de las actividades de formación en las etapas de diseño, el impacto esperado de este trabajo incluye (pero no se limita a):

- La creación de una nueva línea de investigación sobre la relación entre los algoritmos de Machine Learning y los seres humanos respecto a la forma en que ambos aprenden.

- Fomentar el diseño de actividades de formación de una manera medible compatible con análisis y estimaciones.

- Ayudar a mejorar el conocimiento sobre cómo el aprendizaje sucede en realidad.

- Ayudar a encontrar nuevas formas de optimizar el proceso de aprendizaje a través de la estimación adecuada dificultad de las actividades de formación. Esto permite emparejar automáticamente a los estudiantes con actividades de formación adecuadas en la mayoría de los casos.

- Ayudar a la personalización automática del proceso de aprendizaje, proporcionando nuevas herramientas para los Intelligent Tutoring Systems ya existentes.
D.3 Resultados

En este trabajo se han presentado: una definición de dificultad de las actividades de formación como función, una adaptación de esta función para medir la dificultad en los seres humanos, otra adaptación obtener mismo tipo de medida en los algoritmos de Neuroevolución y una definición de similitud entre ambas medidas. El objetivo principal es encontrar una manera de estimar la dificultad de las actividades de formación para los estudiantes en la etapa de diseño de la actividad: antes de poder probarla con estudiantes reales.

Este capítulo se centra en probar estos métodos y definiciones, para validarlos con los resultados reales de los estudiantes. Estas primeras pruebas constituyen una base empírica inicial para estimar la validez de todo el enfoque. Como se trata de una nueva línea completa de investigación, es aconsejable experimentar con los conceptos y obtener algunos resultados empíricos. Entonces, si estos resultados son prometedores, se requerirá más investigación para construir diferentes experimentos, probar diferentes actividades de formación, verificar los diferentes métodos y, con suerte, inducir algún conocimiento teórico. Para ser prácticos, este trabajo lleva a cabo los primeros experimentos con los métodos propuestos, destinados a la validación empírica.

Para lograr los propósitos de este trabajo, se ha diseñado un plan de experimentación compuesto por los siguientes pasos:

- Probar los algoritmos NEAT y HyperNEAT en distintos laberintos de PLMan. Preparar todo el entorno para permitir que un
controlador basado en redes neuronales pueda dar acciones directas a Mr.PLMan. A continuación, poner en marcha experimentos de aprendizaje con ambos algoritmos. Analizar los resultados y realizar una detección precoz de posibles puntos fuertes y débiles.

- Seleccionar los mejores modelos de entrada-salida y conjuntos de parámetros para el aprendizaje. Para ello, se seleccionará un conjunto de laberintos y se llevarán a cabo pruebas de aprendizaje. El foco en este apartado está en ajustar NEAT y HyperNEAT hasta que alcancen la máxima puntuación posible al jugar los laberintos.

- Por último, lanzar una sesión de entrenamiento con modelos y laberintos seleccionados, y comparar los resultados de entrenamiento con los resultados de los estudiantes reales.

D.3.1 Preparación del entorno para la Neuroevolución

Con el fin de preparar todo el entorno para los algoritmos de Neuroevolución, es necesario realizar un pequeño diseño de interfaz de comunicación. Esta interfaz debe permitir que un controlador basado en redes neuronales pueda enviar directamente comandos para controlar Mr.PLMan, de la misma forma que los controladores escritos en Prolog. Este diseño se muestra en la figura 6.1. PLMan está implementado íntegramente en lenguaje Prolog, utilizando SWI-Prolog (Wiemeler, et al., 2012). Los controladores que deciden sobre las acciones de Mr.PLMan se escriben también en lenguaje Prolog, normalmente, e interactúan directamente con el juego a través de la interfaz de pro-
gramación de aplicaciones de PLMan (API). PLMan fue diseñado para operar de esta manera por simplicidad: cualquier persona dispuesta a crear un controlador para PLMan sólo necesita un editor de texto y una primera línea para incluir las definiciones de la API. El resto es puramente programación en Prolog.

Sin embargo, PLMan no fue pensado para ser programado en otros lenguajes. Puesto que los algoritmos NEAT y HyperNEAT originales están implementados en lenguaje C++, se requiere una interfaz de código nativo. Por lo tanto, se ha desarrollado un módulo nativo para SWI-Prolog para cubrir esta necesidad. Este módulo nativo hace uso de la interfaz nativa de SWI-Prolog para proporcionar una interfaz con la que comunicarse con las redes neuronales tanto de NEAT como de HyperNEAT desde un programa Prolog. A través de este módulo, un sencillo controlador en Prolog puede enviar información de entrada a una red neuronal, pedirle que se actualice y obtener una salida en forma de una acción concreta que deba ser realizada por Mr.PLMan (ver figura 6.1).

El diseño mostrado en la figura 6.1 también tiene en cuenta que los controladores de redes neuronales requieren entrenamiento. En este caso, el entrenamiento se lleva a cabo mediante un algoritmo genético que funciona en la capa superior de los algoritmos NEAT y HyperNEAT. Este algoritmo genético produce poblaciones de redes neuronales que se almacenan como archivos de datos. Gracias a esto, los controladores de NEAT/HyperNEAT tienen una única implementación genérica, y su comportamiento cambia conforme se utilizan distintos ficheros de datos como entrada. Esta es la base para el modelo particular de aprendizaje imple-
D.3.2 Ajustes precisos para NEAT y HyperNEAT

NEAT y HyperNEAT son algoritmos Neuroevolución muy potentes que han demostrado ser capaces de enfrentarse con éxito a una gran variedad de problemas de Machine Learning (Clune et al., 2009, Drchal et al., 2009, DAmbrosio et al., 2014, Gallego-Durán et al., 2013, Haasdijk et al., 2010, Hausknecht et al., 2012, 2013, Lee et al., 2013, Lowell et al., 2011b, Yosinski et al., 2011). Sin embargo, este gran potencial viene con un importante coste: ambos algoritmos tienen un montón de parámetros libres que deben ser fijados por los investigadores en tiempo de diseño. Estos parámetros controlan diferentes aspectos de la forma en que estos algoritmos funcionan, y tienen un gran impacto en el resultado final. De hecho, tanto NEAT como HyperNEAT suelen mostrar un rendimiento
bastante bajo cuando son aplicados tal como vienen (con los parámetros por defecto) a nuevos problemas. Esta falta de rendimiento se debe principalmente a una mala configuración de parámetros. Para la mayoría de los problemas, el rendimiento aumenta sustancialmente cuando se establecen los parámetros apropiados.

Desafortunadamente, no existe ninguna metodología definida o forma algorítmica para encontrar el conjunto de parámetros óptimo. La forma más común para establecer los parámetros para este tipo de algoritmos es a través de conocimiento experto sobre cómo funcionan estos algoritmos, y un largo ciclo de ensayo-error. En los experimentos presentados se ha realizado una gran cantidad de trabajo para encontrar un conjunto ajustado de parámetros. En concreto, el método seguido para encontrar un buen conjunto de parámetros ha sido la siguiente:

- Algunos laberintos han sido seleccionados y clasificados con respecto a su dificultad por expertos (profesores). Estos laberintos han sido utilizados para medir el rendimiento de NEAT y HyperNEAT para cada conjunto concreto de parámetros.

- Se ha definido una medida de puntuación o función de meta-fitness. Esta función evalúa el resultado de NEAT y/o HyperNEAT dado un conjunto de parámetros. Las evaluaciones se realizan en función de la puntuación obtenida en todos los laberintos seleccionados y la cantidad necesaria de esfuerzo para obtener la puntuación.

- Se ha seleccionado una forma concreta para recorrer el espacio de parámetros. Teniendo en cuenta que el espacio de parámetros es
enorme, se ha optado por una aproximación voraz. Aunque los algoritmos voraces tienden a estancarse rápidamente en mínimos locales, otras posibles aproximaciones más sofisticadas terminan teniendo un coste exponencial con respecto al tiempo de entrenamiento que requieren. Por lo tanto, aunque puede que la aproximación voraz no sea la mejor en términos de resultado final, con ella obtendremos un resultado final una cantidad limitada y asumible de tiempo.

D.3.3 Proceso de ajuste: selección de laberintos

Los laberintos en PLMan están clasificados en 5 fases (0-4). La fase 0 se llama “fase tutorial”, ya que sólo contiene laberintos que son solucionables con 1 sola línea de código. La intención de estos laberintos es introducir a los estudiantes en PLMan, mostrándoles las acciones Mr.PLMan puede realizar y cómo deben programar y ejecutar controladores en Prolog. Por lo tanto, estos laberintos no son útiles desde la perspectiva del Machine Learning. Todos ellos están descartados ya que a los controladores de redes neuronales les bastaría con aprender a activar un par de las salidas de la red, las correspondientes la única acción y dirección que requiere cada uno de estos laberintos. Esto tendrían que hacerlo sin siquiera tener en cuenta la entrada, por lo que el interés en estos laberintos desde el punto de vista científico es nulo.

Al principio, se pensó en seleccionar todos los laberintos de las fases 1 y 2. Hay un total de 120 laberintos diferentes en estas 2 fases. Estos laberintos incluyen casi todos los mecanismos disponibles en PLMan:
pasillos con los puntos, espacios vacíos, enemigos, objetos, puertas, etc. Por lo tanto, este es un conjunto perfecto en términos de saber si un algoritmo está bien ajustado o no, mirando las puntuaciones obtenidas por el algoritmo en el total de laberintos.

Sin embargo, utilizar un número de laberintos para este propósito resulta ser prohibitivo en términos de tiempo de entrenamiento requerido, ya que tiene que ser multiplicado por la cantidad de parámetros de configuración a probar y las iteraciones de entrenamiento por cada conjunto de parámetros. Por ejemplo, se consideran 67 configuraciones diferentes de parámetros para ser probadas con NEAT, teniendo en cuenta que cada una requiere ser probada 2 veces, lo que hace un total de 134 configuraciones para probar, sólo para NEAT. Con respecto a HyperNEAT, hay 75 configuraciones, lo que hace un total de 150 configuraciones para probar. Eso significa $134 + 150 = 284$ configuraciones. Si esa cantidad se va a probar en 120 laberintos diferentes, hay que lanzar un total de 34080 pruebas. Cada prueba consta de 2000 generaciones de entrenamiento, con un tiempo medio de 0,5 minutos por iteración. Eso haría un total de 65 años aproximadamente. Incluso utilizando 65 ordenadores, un año completo de cálculos sería necesaria sólo para la selección de parámetros, sólo para una estrategia voraz.

Así pues, ha sido necesario realizar una selección con unos pocos laberintos. Sólo 5 de los 120 laberintos han sido seleccionados para la realización del ajuste de parámetros. Estos laberintos se han seleccionado

---

3 los parámetros libres de NEAT y HyperNEAT están explicados en detalle en el apéndice B
tratando de incluir la mayor cantidad posible de la mecánicas de juego, para que la selección resultase representativa. Esto suma una cantidad inicial de tiempo de entrenamiento requerida de 2,70 años aproximadamente. Esta cantidad ha sido reducida mediante el uso de 2 ordenadores simultáneamente y una implementación paralelizada, logrando un tiempo de cálculo final de 0,75 años aproximadamente. Los laberintos seleccionados se muestran en la figura 6.3, mientras que los rangos de parámetros analizados se muestran en la tabla 6.1.

La selección del modelo de entrada está incluida junto con la selección de parámetros, como se muestra en la tabla 6.1. Aunque no deba ser considerado un parámetro propiamente dicho, tanto para NEAT como para HyperNEAT, es en realidad un parámetro del modelo de aprendizaje propuesto. Por lo tanto, se incluye dentro de la estrategia voraz para seleccionar el mejor modelo de entrada, junto con los otros parámetros. El modelo de salida ha sido seleccionada manualmente con anterioridad (ver sección 5.3), ya que ambas opciones propuestas eran lógicas, pero la segunda opción (8 salidas) cuenta un mejor rendimiento, ya que ahorra algunos enlaces a las redes neuronales y, con ello, algunos parámetros también. Esto es mucho más importante para HyperNEAT, debido a que cada neurona de salida eliminada permite ahorrar entre 50 y 1.280 enlaces, dependiendo del modelo de sustratos ocultos seleccionado.

La selección de parámetros incluye además la función de fitness, que también podría ser entendida como un parámetro más, aunque tampoco es propiamente un parámetro. En este caso, la puntuación alcanza mediante un controlador (tal como está definida en la sección 5.4) repre-
sentencias una función de fitness natural para la evaluación de su calidad. Dado que la Neuroevolución pretende generar individuos que obtengan el mayor fitness posible, utilizar la puntuación como valor de fitness hará que este proceso tenga una como objetivo generar controladores enfocados en obtener la máxima puntuación posible.

**D.3.4 Estrategia voraz de ajuste de parámetros**

Llamemos “una ejecución del experimento” a un ciclo completo de los algoritmos de aprendizaje NEAT y HyperNEAT, completando 2.000 iteraciones en cada uno de los 5 laberintos, con un conjunto concreto de parámetros de prueba. Usando esta definición, la estrategia voraz seguida ha consistido en los siguientes pasos:

- **Establecer un conjunto inicial de parámetros ajustados manualmente.** Estos serán utilizados como parámetros iniciales para el algoritmo voraz. Inicialmente se selecciona un conjunto de parámetros similar a los recogidos en la literatura (DAmbrosio, 2011b, Stanley, 2004), ajustándolo mínimamente mediante algunas pruebas y ejecuciones manuales. Estos parámetros iniciales seleccionados se muestran en *cursiva* en la tabla 6.1.

- **Dentro del conjunto inicial de parámetros seleccionados, todos los valores se mantienen constantes, excepto el primer parámetro (Tamaño de la población).** A continuación se realizan seis ejecuciones del experimento para probar cada uno de los valores propuestos para el parámetro Tamaño de la población. Terminadas estas seis carreras, el valor de este parámetro que ofrece el mejor rendimiento
global (como suma de las puntuaciones conseguidas en los 5 laberintos) se utiliza para fijar el tamaño de la población.

- Después de fijar el tamaño de la población, todos los parámetros son constantes otra vez excepto el segundo: Elitismo. De manera similar al tamaño de la población, los 3 valores de Elitismo se prueban en tres ejecuciones del experimento y el mejor resultado se selecciona como valor para el Elitismo.

- Este procedimiento se repite para todos los parámetros.

- Una vez que todos los parámetros han sido fijados una vez con sus mejores valores individuales, el ciclo completo se repite por segunda vez, pero empezando con estos nuevos parámetros recién fijados. Después de este segundo ciclo de refinamiento, el conjunto de parámetros fijos resultante es seleccionado como definitivo.

Los parámetros seleccionados finalmente después de realizar este algoritmo voraz se muestran en la tabla 6.2. Esta selección es la que se ha utilizado para todos los próximos experimentos.

**D.3.5 Resultados de experimentación**

Después de configurar todo el entorno correctamente, el último paso es el lanzamiento de los experimentos diseñados para comparar los resultados de aprendizaje de NEAT y HyperNEAT con los logrados por los estudiantes. Para este propósito, se consideran los 120 laberintos de las fases 1 y 2. Los laberintos de la fase 0 se descartaron por su sencillez: son introductiorios y no aportan mucha información. Los laberintos de
D.3. Resultados

las fases 3 y 4 fueron descartados debido a su complejidad añadida: estos laberintos son indeterministas, por lo que requieren pruebas estadísticas para todos ellos. Esta complejidad está fuera del alcance de este estudio y los resultados finales resultarían mucho más difícil de analizar, debido al ruido introducido por el indeterminismo. Por lo tanto, ya que se trata de una primera validación empírica, mantener los experimentos más sencillos ofrecerá una mejor perspectiva de la validez de los resultados.

Los resultados siguientes proceden de ejecutar NEAT y HyperNEAT utilizando los parámetros de la tabla 6.2 en ejecuciones limitadas a un máximo de 2000 épocas. Además, en caso de que un individuo alcance una puntuación de 100%, la ejecución se detiene pues el algoritmo habrá obtenido la máxima puntuación. Así pues, la ejecución tiene en cuenta el resultado obtenido por el mejor individuo de la población \( \hat{\omega}_t \) en cada época \( t \). Para mantener el modelo lineal seleccionado por \( \hat{E}_t \), sólo se tienen en cuenta las épocas que producen alguna mejora sobre las anteriores. Por ejemplo, para un laberinto dado \( z \), consideremos el escenario de aprendizaje general propuesto por la ecuación 6.1, con una secuencia que incluye a los individuos con mejores resultados. Todos los individuos \( \{ \hat{\omega}_i \mid x < i < n \} \) se eliminan de los resultados finales, ya que no representan ninguna mejora sobre \( \hat{\omega}_x \). Después se utiliza interpolación lineal entre \( S(z, \hat{\omega}_x) \) y \( S(z, \hat{\omega}_{x+n}) \), de la misma forma exacta que es utilizada para los resultados de los estudiantes.

Los resultados completos de todos los experimentos se pueden encontrar en la tabla C.1 (en el apéndice C). Los resultados se dejan en el apéndice para referencia, para hacer este capítulo más claro y fácil de
seguir. En este sentido, dos laberintos representativos han sido elegidos como ejemplo para analizar y discutir. El primero de ellos es el laberinto 1.41 (figura 6.4). Este laberinto está clasificado por profesores como fase 1, dificultad 3. Dado que la dificultad va de 1 a 5, este laberinto es un representante de la dificultad media en la primera fase que los estudiantes tienen que superar. El laberinto contiene una puerta |, una llave a para abrir la puerta y 3 enemigos E (dos que se mueven de arriba hacia abajo y uno que se mueve siguiendo el rectáculo que se encuentra delante de la puerta). Un controlado capaz de superar este laberinto debe esquivar a los enemigos, conseguir la llave, abrir la puerta y comer un total de 43 cocos.

\[ \tilde{E}_{\Theta(t)}(1-41) \text{ y } \tilde{D}_{\Theta(t)}(1-41) \] están representadas en los 2 gráficos superiores de la figura 6.4. Esa parte superior muestra la evolución de la dificultad para NEAT y HyperNEAT conforme ambos aprenden a resolver el laberinto. Los dos gráficos en la parte inferior representan la distancia de NEAT y HyperNEAT con respecto a la dificultad medida en los estudiantes \( D_t(1-41) \). Como se mencionó antes, la semejanza es el complemento de la distancia, por lo que cuanto menor es la distancia, mejor es la estimación de la dificultad respecto a la de los estudiantes actuales. Para este laberinto, NEAT da una estimación relativamente útil para la dificultad real experimentada por los estudiantes. Aunque la similitud no es muy alta (\( \sigma_{2.8}^{NEAT}(1-41) = 0.874 \)), la representación gráfica es útil como una estimación media de los laberintos de fase 1, y

\[ \tilde{D}_{\Theta(t)} \] está representado en los gráficos como \( \tilde{D}_t \) cuando se refiere a NEAT y \( \tilde{D}_t' \) cuando se refiere a HyperNEAT. Esta última notación se mantiene en gráficos que dependen de \( t \) para mayor claridad y simplicidad.
sirve para realizar comparaciones relativas de dificultad con otros laberintos. Sin embargo, HyperNEAT produce una estimación mucho más precisa, con una similitud \( \sigma_{2.8}^{\text{Hyper}}(1-41) = 0.972 \). La comparación gráfica entre los estudiantes y HyperNEAT muestra la calidad de este resultado como predicción sobre la dificultad real.

Como representante de un laberinto más difícil de la fase 2, la figura 6.5 muestra el laberinto 2-11, que ha sido clasificado por los profesores como dificultad 5. Este laberinto tiene 4 enemigos patrulleros \( \text{E} \) con rutas definidas formando cuadrados, 2 estaciones de teletransportarse \( \text{T} \) que teletransportan a Mr.PLMan de un lado al otro y viceversa, 4 arqueros automáticos \( \text{A} \) (que disparar flechas > < cuando ven a Mr.PLMan, y una pistola \( \text{P} \) cargada con 1 bala que puede ser utilizada para matar a un enemigo. Para conseguir superar este laberinto, Mr.PLMan tiene que entrar en la estación teletransportador, comer todos los cocos vigilados por los arqueros automáticos, y volver a la zona principal para terminar de comer el resto de los cocos, hasta el total de 181 cocos.

Los resultados para laberinto 2-11 (figura 6.5) son peores que los obtenidos para el laberinto 1-41 (figura 6.4). Una vez más, HyperNEAT muestra una predicción mucho más precisa de la dificultad real \( \sigma_{16.5}^{\text{Hy} \text{per}}(1-41) = 0.865 > \sigma_{16.5}^{\text{NEAT}}(1-41) = 0.737 \). Sin embargo, esta vez la predicción realizada no puede considerarse suficientemente precisa, ya que una similitud de 86.5% equivale a una diferencia del 13.5%, que puede ser demasiado alta dependiendo del contexto. Sin embargo, la representación gráfica de los resultados sigue siendo muy interesante. Ambos algoritmos de Neuroevolución muestran similitudes de comportamiento interesantes.
con respecto a los estudiantes. Por ejemplo, ambos muestran que la entrega de un controlador capaz de comer \(\approx 50\%\) de los cocos es relativamente fácil y rápida de hacer. Esto es algo que los estudiantes también muestran en su gráfico de progreso. Es importante tener en cuenta que los estudiantes no empiezan de cero, puesto que ya han completado entre 8 y 10 laberintos cuando se enfrentan al laberinto 2-11. La mayoría de ellos ya han desarrollado controladores para laberintos anteriores que con sólo cortar y pegar les sirven para obtener algunos buenos resultados iniciales, con un esfuerzo mínimo. Esa es la razón de la curvatura inicial de su gráfico de dificultad. Por el contrario, la Neuroevolución comienza desde cero cada vez, lo que es una seria desventaja. Por lo tanto, observando la similitud de las curvaturas se obtiene un resultado verdaderamente interesante que plantea nuevas preguntas, ¿Podrían haber sido los resultados más similares si los estudiantes hubieran comenzado desde cero o, al menos, con menos experiencia previa?

### D.3.6 Reflexiones sobre los resultados globales

NEAT y HyperNEAT muestran diferentes habilidades y maneras de encontrar controladores de redes neuronales capaces de resolver laberintos de PLMan. De hecho, la configuración seleccionada para NEAT demuestra ser generalmente más rápido en la búsqueda de controladores válidos (es decir, que necesita muchoas menos épocas). Sin embargo, parece que tiene gran tendencia a quedarse atascado en algunas soluciones concretas, lo que es muy similar a estar atrapado en mínimos locales. HyperNEAT muestra ritmo más lento en encontrar buenos controladores, pero
con mejor capacidad de salir de mínimos locales. Estas diferencias entre NEAT y HyperNEAT resultan cruciales para el propósito de este trabajo, ya que gracias a ellas los resultados son mejores de lo esperado. El ritmo más lento de HyperNEAT se correlaciona mucho mejor con el ritmo de los estudiantes, configurándolo como un estimador decente de la dificultad de aprendizaje de los estudiantes. Los resultados cuantitativos completos se muestran en la tabla C.1 para referencia y posterior análisis.

La pregunta más interesante planteada al inicio de este trabajo cuestionaba la existencia de una correlación entre la Neuroevolución y los estudiantes con respecto a su progresión de aprendizaje cuando se enfrentan a la misma actividad de formación. La existencia de esta correlación supondría que ambas funciones de dificultad resultasen comparables, por lo que los resultados generados con neuroevolución resultarían válidos para la finalidad de estimación. Para responder a esta pregunta, el resultado más importante es la distribución de las similitudes calculadas para todos los laberintos de prueba. Por lo tanto, teniendo en cuenta $Z$ como el conjunto que contiene todos los laberintos en los que NEAT y HyperNEAT han sido analizadas ($|Z| = 120$), las similitudes se distribuyen como indica la ecuación 6.2.

La figura 6.6 muestra estas distribuciones claramente. Esto confirma que efectivamente existe una correlación entre la función dificultad para NEAT/HyperNEAT y la de los estudiantes. Si no existiera esta correlación, los resultados estarían distribuidos de manera uniforme, dando máxima entropía. El hecho de que ambas similitudes se distribuyan según una normal tiene un significado práctico: dado un laberinto al
azar \( z \in Z \), y los resultados de entrenamiento de NEAT o HyperNEAT al resolver \( z \), los eventos más probables son \( \sigma^\text{NEAT}_t(z) = 0.845 \) and \( \sigma^\text{Hyper}_t(z) = 0.885 \). Por lo tanto, después de entrenar NEAT o HyperNEAT y calcular su función dificultad para un determinado laberinto \( z \), es factible asumir que la función dificultad obtenida tendrá una similitud de 0.845/0.885 con la de los estudiantes, dependiendo del algoritmo seleccionado. Este resultado puede ser utilizado para crear un predictor basado en NEAT/HyperNEAT, estableciendo un intervalo de confianza sobre los resultados obtenidos.

La consideración más importante de estos resultados es que son una primera validación empírica para el método propuesto. Es importante ser escrupuloso, ya que esta investigación tiene limitaciones y se centra en una actividad concreta (el juego PLMan) y con un relativamente pequeño subconjunto de laberintos (120) y estudiantes (336). Los resultados son importantes en el sentido de que son un primer paso positivo para atraer más investigación hacia el campo. Confirman que para este caso concreto y subconjunto existe la correlación, que sin duda es valiosa. Sin embargo, la generalización de los resultados aún no es posible: se requiere más investigación para demostrar si se pueden obtener resultados teóricos generales.

Otra consecuencia interesante de todo el sistema diseñado reside en los gráficos de dificultad como potente herramienta de análisis. Como se ha demostrado durante los análisis anteriores, incluso cuando la similitud es baja para un laberinto concreto, las curvaturas de la función dificultad transmiten mucha información sobre el laberinto. Las curvaturas
muestran partes fáciles y complicadas del laberinto, problemas potenciales, así como claves acerca de la relación entre el comportamiento de la Neuroevolución y el de los estudiantes, en su manera de encontrar un controlador válido. La sección C.2 proporciona ejemplos adicionales junto con sus análisis para mostrar más detalles sobre la importancia de este resultado.

También es interesante mencionar un par de casos especiales. Como se puede observar en las tablas C.3 y C.4, hay dos laberintos sin valor de similitud: 2-06 y 2-31. Ambos laberintos pueden verse en la figura 6.7. Su estructura es similar: los cocos están ocultos detrás de un mecanismo en los dos casos. El laberinto 2-06 tiene una portería (|) que sólo puede ser destruida lanzando pelotas (o) y anotando 3 goles. Las pelotas aparecen constantemente a 2 celdas de distancia del marcador. El laberinto 2-31 tiene una puerta (]) que tiene que ser abierta con una llave (\&). En ambos casos, el problema es el mismo para la Neuroevolución: para resolver el laberinto es necesario entender algunos conceptos y trazar un plan de acción. Por otra parte, no hay cocos que puedan guiar a un controlador a través del incremento en su fitness. Por lo tanto, incluso si algunos individuos realizan parcialmente algunas acciones requeridas, su fitness siempre será 0, lo mismo que los individuos que no hacen nada o cometen errores. Esto detiene la evolución, impidiendo el aprendizaje a través de algoritmos genéticos y/o Neuroevolución. El tratamiento para estos casos especiales queda fuera del alcance de este trabajo, y es materia de discusión para la investigación futura.

Los resultados presentados confirman que la forma en que los algo-
ritmos probados de Neuroevolución aprenden a resolver laberintos de PLMan está correlacionada con la forma en que lo hacen los estudiantes. Este es un resultado muy importante, en el sentido de que confirma que hay similitudes entre ambas formas de aprendizaje. En consecuencia, la Neuroevolución podría ser utilizada para predecir la dificultad de actividades de formación en las etapas de diseño, anterior a cualquier prueba hecha con estudiantes reales.

Sin embargo, también es importante destacar las limitaciones de estos resultados. Los experimentos llevados a cabo son sólo una confirmación empírica, por lo que sólo se aplican al conjunto de datos utilizado. Aunque parece ser probable que estos resultados se repitan en otras actividades de formación similares, los resultados de este trabajo no constituyen una prueba de ello. Como no se ha realizado ninguna prueba teórica, se deben ejecutar nuevas series de experimentos similares para diferentes conjuntos de datos, si se quiere disponer de pruebas empíricas que validan este enfoque particular. Por otra parte, aunque los resultados pueden ser utilizados como predicciones directas, su precisión debe ser considerada con cuidado. Lo más probable es que un uso directo de los resultados como predicciones tenga una utilidad limitada en un escenario real. Como se dijo antes, es muy recomendable incorporar un intervalo de confianza a los resultados que, teniendo en cuenta las desviaciones estándar de ambas distribuciones, deberá tener un tamaño considerable. Dependiendo del escenario de aplicación, podría llegar a suponer un problema.

Finalmente, los resultados presentados en esta sección validan la hipó-
D.4. Conclusiones 195

D.4. Conclusiones

Este trabajo partió de la hipótesis de que es posible utilizar el coste de entrenamiento de un algoritmo de Machine Learning para estimar el coste de aprendizaje para los seres humanos. Más concretamente, se suponía que era posible encontrar una nueva definición de dificultad que permitiera medirla a partir de datos experimentales de estudiantes reales, y estimarla utilizando Machine Learning.

D.4.1 Contribuciones aportadas

A partir de la hipótesis, se ha desarrollado y presentado una definición general de dificultad. Esta nueva definición ha sido diseñada sobre la base de una lista de propiedades deseadas que fueron enunciadas previamente. Mediante el uso de esta nueva definición propuesta, la dificultad es medible, se puede comparar y visualizar, y se está relacionada con el esfuerzo realizado en el tiempo. El esfuerzo queda modelado como el tiempo re-

---

5Enunciada en la sección 1.1
querido para conseguir una puntuación concreta. Así pues, la definición propuesta de dificultad tiene en cuenta el progreso hacia la resolución de una actividad de formativa, basándose en la puntuación que un agente logra mientras realiza la actividad.

La definición propuesta de dificultad tiene limitaciones en el sentido de que las actividades tienen que cumplir algunos requisitos para ser medibles a través de esta definición:

- Deben ser realizables en el tiempo\(^6\).
- Debe haber una función de puntuación que mida el progreso. Esta función, además, debe tener límites superior e inferior y ser no-strictamente creciente. Es decir, no debe ser posible perder puntos con el tiempo.

Esta definición propuesta tiene también muchas ventajas interesantes, la mayoría de las cuales provienen de su propia condición de función: al contrario que si fuera un valor adimensional simple, tiene la capacidad de mostrar el progreso en el tiempo. Su representación gráfica muestra características del estudiante y la actividad de formación de manera visual. Gracias a eso se pueden identificar diferentes partes de las actividades de formación. Por ejemplo, las partes más difíciles producirán valles en el gráfico, permitiendo tanto su identificación como su medición. Estas gráficas de dificultad permiten comparar actividades, generando un conocimiento mucho más detallado sobre cuáles requieren más esfuerzo,

\(^6\)Esto es, no se trata de actividades instantáneas. Un ejemplo de una actividad instantánea sería dar una respuesta a una pregunta de 4 opciones, teniendo en cuenta sólo la respuesta dada y no el tiempo necesario para pensarla.
así como las diferencias en la distribución del esfuerzo en el tiempo. Estas ventajas hacen de la definición propuesta de dificultad una potente herramienta para el análisis y la comparación de las actividades de formación.

Como actividad específica que utilizada para la experimentación empírica se ha presentado un juego llamado PLMan. El juego está siendo utilizando actualmente en la Universidad de Alicante para enseñar programación en Prolog, Lógica y una breve introducción a la Inteligencia Artificial. Se ha demostrado que PLMan cumple con las propiedades requeridas para aplicar la definición de dificultad. También se ha demostrado su potencial como clase genérica de actividades de formación: los estudiantes resuelven distintos laberintos de PLMan y cada laberinto es una actividad de formación en sí misma. Por lo tanto, PLMan es una clase genérica de actividades con contenidos similares pero diferentes dificultades de aprendizaje.

También se ha mostrado que muchos juegos tienen un parecido considerable con PLMan en sus reglas principales. Esta característica hace que todos estos juegos puedan adaptarse fácilmente a la definición propuesta de dificultad. Por lo tanto, también podrían ser considerados como actividades medibles y estimables con los métodos propuestos en este trabajo.

Se ha creado una versión de la función de dificultad específicamente adaptada para PLMan. Esta versión se ha utilizado para medir la dificultad en 220 laberintos diferentes que han sido realizados por 336 es-
tudiantes\footnote{A cada estudiante se le asigna algunos laberintos al azar del total disponible. En función de su desempeño, cada estudiante puede llegar a tener entre 8 y 18 laberintos asignados en total.}. Algunos resultados relevantes de estas mediciones han sido presentados y explicados en detalle.

Para continuar con los objetivos de esta investigación, se ha seleccionado la Neuroevolución como método de Machine Learning para aprender a resolver automáticamente laberintos de PLMan. Tras revisar el estado del arte en el área, se han seleccionado dos algoritmos en concreto: NEAT y HyperNEAT. Con ellos se han diseñado los modelos específicos para las redes neuronales. Concretamente, se han propuesto varios modelos de entrada y salida tanto para NEAT como para HyperNEAT, además de algunos modelos de sustratos en capa oculta para HyperNEAT.

Por otra parte, se ha propuesto una definición adaptada de dificultad para obtener medidas para NEAT y HyperNEAT. Las medidas obtenidas con esta definición tienen que ser comparadas con las obtenidas a partir de los datos de rendimiento de estudiantes reales. Por lo tanto, también se ha diseñado una función de similitud a partir de ambas funciones de dificultad. Esta función similitud mide el complemento a la zona comprendida entre las dos curvas de dificultad. Dado que el área comprendida entre las dos curvas corresponde a la diferencia entre ambas funciones, su complemento mide la similitud, tal como se deseaba.

Se ha desarrollado una adaptación para PLMan con la intención de realizar los experimentos de aprendizaje con Neuroevolución, poder obtener sus costes de entrenamiento y medir su similitud con los estudiantes. Esta adaptación permite que Mr.PLMan pueda ser controlado por una red neu-
ronal en lugar de una base de conocimientos Prolog. Una vez hecha esta adaptación, se ha procedido a ajustar NEAT y HyperNEAT probando conjuntos de parámetros de configuración siguiendo una aproximación voraz. A partir de estas pruebas de ajuste, se han seleccionado los conjuntos de parámetros que obtenían mejores resultados de aprendizaje.

Una vez seleccionados los conjuntos de mejores parámetros, NEAT y HyperNEAT han sido entrenados para resolver 120 laberintos PLMan de las fases 1 y 2. Los resultados obtenidos han dado una medida del coste de aprendizaje para NEAT y HyperNEAT. Estos resultados han sido comparados con los de los estudiantes utilizando la función de similitud definida previamente. Con esta última comparación se han obtenido 240 valores de similitud: 120 sobre los costes de aprendizaje de NEAT con respecto a los de los estudiantes, y otros 120 sobre HyperNEAT con respecto a los estudiantes. Los resultados son muestran que los valores de similitud entre NEAT y los estudiantes se distribuyen normal con media 0,845 y desviación estándar 0,130. Por su parte, los valores de similitud entre HyperNEAT y los estudiantes también se distribuyen normal con media 0,885 y desviación estándar 0,126. Estos resultados confirman la hipótesis de partida de este trabajo: en efecto, existe una correlación entre los costes de aprendizaje para NEAT/HyperNEAT y los de los estudiantes. Para ser exactos, estas correlaciones existen en el contexto de los experimentos realizados y con el conjunto de datos utilizado.

Además de la validación de la hipótesis principal, a continuación se resumen todas las contribuciones que han sido presentadas en este trabajo:
• Una nueva definición de dificultad con propiedades interesantes para analizar el progreso de los estudiantes en la resolución de actividades de formación.

• El juego PLMan como actividad de formación y como modelo para adaptar fácilmente otras actividades de formación al cumplimiento de los requisitos de la definición propuesta de dificultad.

• La aplicación de la Neuroevolución como algoritmo para aprender a resolver laberintos PLMan automáticamente.

• Una función de similitud para medir la precisión al estimar los costes de aprendizaje de los estudiantes a través de los costes de entrenamiento de NEAT y/o HyperNEAT.

• Una nueva aplicación de la Neuroevolución para estimar la dificultad de las actividades de formación en etapas de diseño.

Por otra parte, la hipótesis principal ha sido validada demostrando que existe una correlación entre los costos de entrenamiento de algoritmos de Neuroevolución y los costes de aprendizaje de los estudiantes. Por lo tanto, la dificultad que los estudiantes experimentan al resolver un laberinto de PLMan, podría ser estimada utilizando la función de dificultad que se obtiene al medir los costes de entrenamiento de NEAT o HyperNEAT para ese mismo laberinto.

Siendo escrupulosos, es importante señalar que la precisión de la estimación no es muy alta. Sin embargo, aplicando un intervalo de confianza la estimación es válida como una primera aproximación sobre la dificultad que experimentarán los futuros estudiantes al resolver laberintos de
PLMan. Por otra parte, la comparación entre las diferentes funciones de dificultad resultantes de los algoritmos de Neuroevolución también se pueden utilizar como comparaciones estimativas de las dificultades reales para los estudiantes. Esta última aplicación sería probablemente mucho más precisa que los valores absolutos de dificultad para cada laberinto.

Las aportaciones presentadas en este trabajo abren un nuevo campo de investigación: el uso del Machine Learning como estimación del modelo de aprendizaje humano.

**D.4.2 Trabajos futuros**

Como se ha mencionado en la sección anterior, uno de los resultados más interesantes de este trabajo es la apertura de un nuevo campo de investigación relacionado con el propio aprendizaje, con las diferentes formas de aprendizaje (de los seres humanos frente al Machine Learning) y las correlaciones existentes. Este campo puede dar resultados interesantes en futuras investigaciones, ya que tiene potencial para producir nuevos descubrimientos en el camino del aprendizaje en los seres humanos. Por otra parte, el circuito de retroalimentación que genera también tiene potencial para impulsar el desarrollo de nuevos algoritmos de Machine Learning y mejorar los ya existentes en la base de su similitud con el aprendizaje humano. Estas son las líneas interesantes que serán un reto para explorar, resultando muy gratificante si alguna de ellas se materializa finalmente.

Sin embargo, estas posibilidades probablemente requerirán muchos pasos previos y años de investigación. Siendo prácticos, estos son caminos inmediatos que esta investigación debería seguir:
• Desarrollar una nueva manera de entrenar NEAT/HyperNEAT con el objetivo de obtener un rendimiento más cercano al de los seres humanos. En este trabajo, ambos algoritmos han sido entrenados de la forma tradicional, tratando de lograr el máximo rendimiento en el sentido de capacidad para resolver laberintos PLMan. Sin embargo, ¿Sería posible utilizar las funciones de similaridad como entrada para la función de fitness? ¿Hay alguna otra manera de entrenar ambos algoritmos para resolver laberintos PLMan de la forma más parecida posible a como lo hacen los seres humanos? ¿Puede esto codificarse en los objetivos de ambos algoritmos de Machine Learning?

• Exportar este método para otras actividades de formación. Encontrar correlaciones similares en actividades de formación diferentes a PLMan para poder confirmar un patrón más general y poder continuar con el desarrollo de resultados teóricos. Muchas actividades y experimentos adicionales serán necesarios para poder obtener resultados en esta línea.

• Proponer una hipótesis teórica inicial sobre la existencia de una correlación general entre el Machine Learning y el aprendizaje humano. ¿Podría haber una similitud general entre todos los posibles modelos de aprendizaje? Esta línea representa un trabajo a muy largo plazo, pero el objetivo final podría ser de gran relevancia, tanto en Inteligencia Artificial como en ciencias sociales (cualquier campo relacionado con el aprendizaje humano o el aprendizaje en general). Aunque es pronto para empezar esta línea de
investigación, es importante tenerlo en cuenta, ya que a menudo las ideas teóricas alimentan experimentos prácticos y generan ciclos de retroalimentación muy útiles para el progreso de la investigación.
Bibliography


Genetic and evolutionary computation, GECCO ’08, New York, NY, USA, pp. 819–826. ACM.


Kormushev, P., S. Calinon, and D. G. Caldwell (2013). Reinforce-


Lowell, J., S. Grabkovsky, and K. Birger (2011a, Aug). Comparison of neat and hyperneat performance on a strategic decision-making prob-


Mendes-Moreira, J. a., C. Soares, A. M. Jorge, and J. F. D. Sousa (2012,


