Heriot-Watt University

Masters Thesis

Immune-based training against the Level-k reasoning model in the Keynesian Beauty Contest

Author:
Alexandre KHA

Supervisor:
Dr. Wei PANG

A thesis submitted in fulfilment of the requirements for the degree of MSc.

in the

School of Mathematical and Computer Sciences

September 2023

Declaration of Authorship

I, Alexandre KHA, declare that this thesis titled, 'Immune-based training against the Level-k reasoning model in the Keynesian Beauty Contest' and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Alexandre KHA

Date: 16/04/2023

"Prediction is very difficult, especially about the future."

Niels Bohr

Abstract

This research project aims at developing an immune-trained agent, able to play the game known as the Keynesian Beauty Contest (KBC). In this game, many players choose during several rounds an integer in the range $[\![0, 100]\!]$, with the objective of being the closest to the average of all choices times the factor $2/3$.

Artificial Immune Systems are a class of metaheuristics inspired by the mammalian immune system. Its applications include function optimisation, which makes them appealing to "evolve" optimal decision processes in the KBC. Our framework is inspired by [Nagel, 1995], proposing that players have a decision pattern based on the Level-k model. Thus, we want to optimise an agent that has to predict how to beat crowds of players, simulated using Level-k patterns. Training is done by confronting the agent against all possible crowds, while it does not know *a priori* other players' decision pattern.

Two response models were implemented. The first one optimises the values of a dictionary, representing responses to a previous average. The second one is Neuroevolution, which models the decision process as an Artificial Neural Network, and updates its weights with a metaheuristic, rather than with Backpropagation. The designed input is either the memory of the last average (memory-1 case) or the two last ones (memory-2 case), while the output is a prediction. Both models are optimised using the immune algorithm CLONALG, and the Genetic Algorithm (GA) for comparison.

Experiments show that CLONALG succeeds to reach, in reasonable time, very good and slightly better performance than GA, of notably 76% win rate for the memory-1 neuroevolved model. An hybrid model combining the plays of memory-1 and memory-2 networks increases this success rate to 81%, which can be considered excellent.

Acknowledgements

I would like to thank Dr. Wei Pang for his availability, thoughtful guidance, and advice throughout this research project.

Contents

List of Figures

List of Tables

Abbreviations

KBC	**K**eynesian **B**eauty **C**ontest
PD	**P**risoner's **D**ilemma
IPD	**I**terated **P**risoner's **D**ilemma
AIS	**A**rtificial **I**mmune **S**ystem
NSA	**N**egative **S**election **A**lgorithm
CS	**C**lonal **S**election
AINET	**A**rtificial **I**mmune **NET**work
GA	**G**enetic **A**lgorithm
RWS	**R**oulette **W**heel **S**election
TS	**T**ournament **S**election
ANN	**A**rtificial **N**eural **N**etwork
CNE	**C**onventional **N**euro**E**volution
NEAT	**N**euro**E**volution of **A**ugmenting **T**opologies
UAV	**U**nmanned **A**erial **V**ehicle
HPC	**H**igh-**P**erformance **C**omputing
WC	**W**in **C**ount

Symbols

$f(.)$	continuous function
$f[.]$	discrete function (or list)
$[\![A, B]\!]$	Range of integers between A and B
P_i^k	Player i of rationality level k
S^N	If S is a set, the N-times Cartesian product of S with itself
$\bar{X}$ *or* $Avg(X)$	Average of the set X
$\sigma(.)$	sigmoid function, defined by $\sigma(x) = \frac{1}{1+e^{-x}}$
$\tanh(.)$	hyperbolic tangent function, defined by $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
$reLu(.)$	Rectified Linear Unit function, defined by $reLu(x) = max(0, x)$
$\mathcal{U}\big([a, b]\big)$	Uniform distribution on the real interval $[a, b]$
$\mathcal{N}(\mu, \sigma^2)$	Gaussian (Normal) distribution, centered on μ and of std. σ
$\mathcal{B}(p)$	Bernoulli distribution : $P(X = 1) = p$ and $P(X = 0) = 1 - p$
$\mathcal{G}(p)$	Geometric distribution : $P(X = k) = (1 - p)^{k-1} \times p$
$\binom{n}{k}$	Number of k-combinations in a set of n items (Binomial coefficient)
$\lfloor . \rfloor$	Floor function
$\lceil . \rceil$	Ceil function
$Round(.)$	Rounding-to-nearest-integer function
$\equiv$	Congruent or identical symbol

Chapter 1

Introduction

Artificial Immune Systems (AIS) find their source of inspiration in the Mammalian Immune System, in order to find solutions to problems, which can be as diverse as data mining [Timmis et al., 1999], computer security [Jungwon and Bentley, 2002] or optimisation [Coello and Cruz Cortes, 2002]. Like other biology-inspired techniques, AIS can be a powerful tool to solve problems, especially when known methods fail or are not applicable to solve them. Genetic Algorithms (GAs) and Particle Swarm Optimisation (PSO) are certainly among the most popular biology-inspired techniques for optimisation. AIS are less used in general, if we compare the literature volume of these fields. However, AIS have proven to give a good metaphor for optimisation as well as good performances depending on the applications. In relatively recent years, some publications have been made about AIS in regard to general optimisation [Li et al., 2021, Savsani et al., 2014], and their results are encouraging to still consider them as a powerful optimisation tool.

Decision taking is particularly related to the problematic of optimisation, since the objective is often to either increase the resulting gain of a decision, or minimise its loss or cost. It is also one of the main topics of Game Theory - a popular field having applications notably in Economics [Gibbons, 1992] - which investigates decision taking and behaviours dynamics in game contexts. We may note that while some GAs have been applied to tackle decision taking problems, for instance in Robotics or Game Theory - subjects further expanded in Chapter 2 - AIS are very less investigated in general.

Therefore, in this paper, we investigate the possibility to build an immune-trained agent able to solve consistently a problem issued from Game Theory, known as the Keynesian Beauty Contest (KBC). This well known game finds its origin in the analogy made by the economist Keynes [1936] between a beauty contest and stock market fluctuation. Many variants of this game exist, and one particularly popular - which is inspired by Moulin [1986] - is the following: N players choose an integer between 0 and 100 with the objective of being the closest to the average of all players, times a proportion $p = 2/3$. In an iterated version, which interests us in this paper, players have the ability to modify their choice in function of previous results, and therefore adapt to their opponents. Our framework sets the number of rounds at 10, and the number of players at 5. The challenge of this paper is to develop an immune-trained agent which adapts its strategies to a dynamic environment, the opposing players. This environment is simulated to follow the Level-k rational patterns, a model proposed by Nagel [1995]. It states that each player has a rationality level k, characterised by playing the previous average times p^k ($Level_k[t] = Avg[t-1] \times p^k$).

We may note that our framework is somewhat specific and does not aim at solving the game universally. It does not aim to play against real players either, who can presumably have more diverse playtypes than the Level-k model. Rather, it considers the Level-k model as a dynamic benchmark to optimise - or more specifically to anticipate - and the project is mainly a proof of concept concerning the question of building decision-taking agents with AIS. However, when finding optimal strategies, the agent might give more insight on the nature of the game, and ideally would extrapolate good plays against real players if Nagel's model is accurate. It is a topic that would need to be investigated in a further work.

Following Literature Review, presented in Chapter 2, two solutions are investigated in this study. The first one is quite rudimentary and consists in evolving the outputs of a dictionary, which keys are the possible values of the last memory (previous average). The second one is Conventional NeuroEvolution (CNE), which uses a metaheuristic to adjust without supervision the weights of a neural network, as it plays against groups of opponents. Both models are optimised using an immune-inspired algorithm named CLONALG, as well as GA for comparison.

Chapter 2

Background and Literature Review

2.1 Characteristics of the KBC and the Level-k model

2.1.1 Illustration

We illustrate in Figure 2.1 the process of the KBC, inspired by Keynes [1936] and reformulated by Moulin [1986]. Players within a group have to guess the average of the numbers chosen by the others, times $p = 2/3$. Moulin originally proposed the set of possible choices as $[\![1, 999]\!]$; however, it is not unusual to see this range being rather restricted between 0 and 100 for study simplification by other authors [Ho et al., 1998, Nagel, 1995].

In the iterated KBC, players are announced individually at each end of round, the average of the group, the actual target and whether or not they have won the round. They don't know the others' choice, which prompts them to base their decision on the global behaviour of the crowd for the next round.

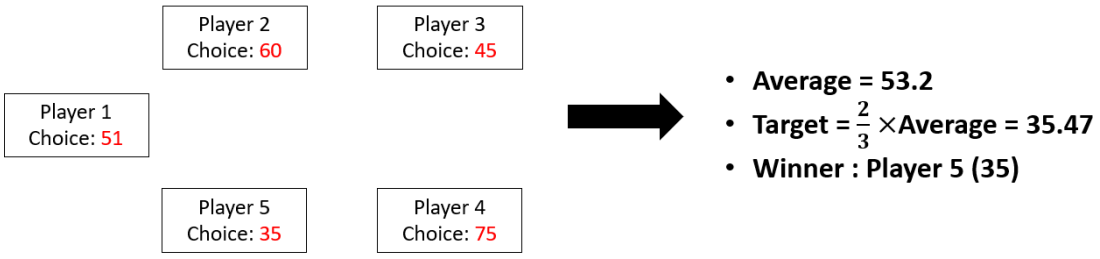


FIGURE 2.1: Example of a KBC round, between 5 players, with the parameter $p = 2/3$

2.1.2 Nash Equilibrium and Iterated Dominance

It is relevant to wonder whether there exists an optimal play in the KBC or not. There indeed exists one, but in the sense of the Nash Equilibrium, which is defined as follows: once the Nash Equilibrium is reached, any change of strategy while other players keep theirs is sub-optimal in terms of payoff (or reward). We can translate this definition mathematically, by considering a game with $N \geq 2$ players, having - to simplify - at any step of the game a set of finite strategies S (in the KBC, $S = [\![0, 100]\!]$). Let us also define a payoff function for each player $u_i : S^N \to \mathbb{R}$, rewarding or penalising i depending on the comparison between their strategy and the other players'. Then, a Nash Equilibrium (if it exists) is a state $s = (s_1, s_2, ..., s_N) \in S^N$ which verifies [Fabrikant et al., 2004]:

$$\begin{gathered} \forall i \in [\![1, N]\!],\ \forall s'_i \neq s_i \in S, \\ u_i(s_1, ..., s_i, ..., s_N) \geq u_i(s_1, ..., s'_i, ..., s_N) \end{gathered} \tag{2.1}$$

In the KBC, the Nash Equilibrium is the state where every player chooses 0. Indeed, this equilibrium can be obtained through iterated dominance, which consists in iteratively eliminating worse (strictly dominated) solutions relying on assumptions about other players' behaviour [Ho et al., 1998]. To illustrate, when applying dominance 1 time in this game, one may notice that if everybody chooses the maximum integer (100), the target becomes 67. So assuming everybody else had this reasoning, the player may realise it is futile to choose any integer greater than 67. But expecting everybody to choose below 67, we could dominate the possible range of integer one more time, which would lead to conclude we should not choose any number greater than 45 (0.67×67), and so on. If everyone anticipates each other perfectly, they will all converge to 0, which

is therefore the Nash Equilibrium. Everybody wins by playing 0 and no other strategy can win if others keep playing 0.

However, even though there exists a situation where nobody gains in playing anything else than the Nash Equilibrium, playing 0 doesn't ensure to win every time. Indeed, everybody has to anticipate each other perfectly to play the Nash Equilibrium. For example, if one chooses 0 while the others (in a crowd of 5 players) choose 30, the target is $\frac{2}{3} \times \frac{0+4\times30}{5} = 16$, and the latter players are therefore closer to it. In some games like the Traveler's Dilemma, the Nash Equilibrium has a seemingly paradoxical value because it rewards a weak payoff, while other strategies - more risky - reward more [Basu, 1994]. This explains why the Nash Equilibrium is not guaranteed to be observed in sociological experiments. Reaching the Nash Equilibrium does still prove a long-term understanding of the game, and observing it should attest a certain quality of response of an agent in our framework.

2.1.3 The Level-k model

The KBC is not trivially solved by playing 0, since all players have to anticipate each other perfectly to reach the Nash Equilibrium. Some researchers made sociological experiments and tried to elaborate models of playing which people tend to adopt. It is the case of Nagel [1995], whose experiments tend to validate the Level-k model of reasoning. In this model, a level-k player plays by considering each other to be of level-$k-1$. Applied to the KBC, a level-k player chooses the previous average, and multiply it by p^k ($[\frac{2}{3}]^k$ in the Moulin variant we adopt). Nagel did find out that people's rationality level very rarely exceeded 3, and didn't seem to change over time. In a framework which lasts for 4 rounds, the optimal play was often close to a level-2 strategy. Even though her experiments were quite conclusive, some gaps with the model were sometimes observed, which indicates that it is not perfectly accurate. However this model constitutes the basis of our research, which would consist in optimising the number of wins possible by playing against k-leveled simulated players. Therefore, we illustrate in Figure 2.2, the iterative KBC workflow in which the agent would evolve. It plays during, say 10 turns against a crowd of $N = 5$ simulated players with a given rationality level, and we want it to win the most rounds possible. The agent is assumed to rely on $Avg[t-1]$ and

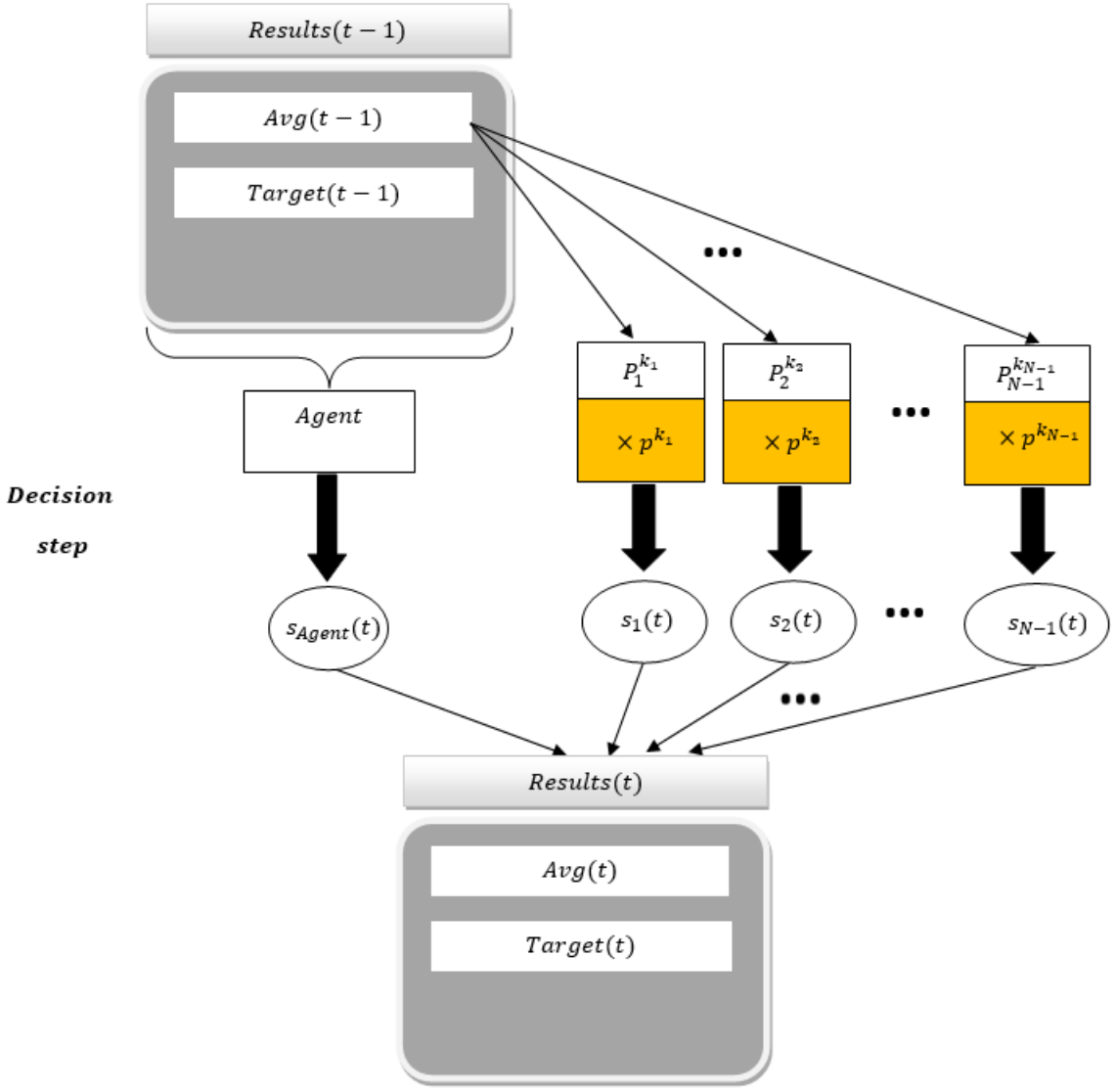


FIGURE 2.2: Iterative KBC workflow. Adopted parameters are: $N = 5$, $p = 2/3$, $k_i \in \{0, 1, 2, 3\}$, $t \in [\![1, 10]\!]$

$Target[t-1]$ to play, but actually only one of these values would be significant since they are proportional by a factor $p = 2/3$. The player i of rationality k_i is noted $P_i^{k_i}$.

2.2 Mutation-based metaheuristics: the GA and AIS

2.2.1 Presentation of the Genetic Algorithm (GA)

The GA is a metaheuristic - i.e. is a stochastic search method - aiming to find solutions to optimisation problems. It builds solutions drawing inspiration from the Theory of Evolution formulated by Darwin [1831]. In this metaphor, the quality of a candidate

solution, called chromosome and often represented by a vector, is measured by a fitness function, which we consider by convention best when it has high value. Over time steps, called generations, new solutions are built using genetic operations, like natural selection of parents, genome mixing by crossover (reproduction) and mutation of the genes (the components of the chromosome). It has proven to be particularly efficient on non-trivial problems where classical methods fail or are not applicable due to inconvenient properties of the fitness function (lack of differentiability, high-multimodality ...) [Kramer, 2017]. Among an exhaustive list of applications, we can mention planning and scheduling, and design engineering [Man et al., 1996], a notable example being the conception of an antenna shape for optimal wave transmission by NASA [Lohn et al., 2008]. In regard to our problematic, the GA is a priori a suitable technique which would typically be used to optimise the performance, thought as the win count, of the agent against level-k players.

2.2.2 Presentation of Artificial Immune Systems (AIS)

AIS are another large class of metaheuristics, often distinguished from the GA even though some similarities exist between them. In contrast to the GA, AIS uses the metaphor of the mammalian immune system to find solutions to a problem. This field has particularly generated interest in the 90's, when huge progress has been done in Immunology [Brownlee, 2011]. AIS gather 4 main paradigms drawing inspiration from different aspects of the immune system, and which have different aims. Two of them, the Negative Selection Algorithm (NSA) and Danger Theory, are more concerned with classification and anomaly detection, investigating the notions of self (and non-self) cells [Forrest et al., 1994] and of danger signal [Greensmith et al., 2005] to trigger detection. These are interesting topics, which are however not directly related to our problematic.

The two last artificial-immune paradigms, called Clonal Selection (CS) and Artificial Immune Network (AINET) are however concerned with fitness optimisation like GA, which relates directly to our subject. CLONALG, introduced by De Castro and Von Zuben [2002], draws inspiration from the clonal selection of lymphocytes - indifferently called antibodies here - to counter antigenic agents. We could say that CLONALG repeats practically the same steps as GA, except that there is no crossover between parents but

cloning, and that the mutation - called hypermutation - has a varying realisation probability which decreases the fitter solutions become. In this paradigm, a candidate solution is also called antibody, instead of chromosome. The AINET, or rather opt-AINET, as introduced by its inventors De Castro and Timmis [2002], can be seen as an alternative of CLONALG, where however, inhibiting interactions exist between antibodies, constituting a network. Indeed, taking inspiration from the Immune Network Theory [Jerne, 1974], it maintains diversity by removing the antibodies which are too similar, often leading to detect a set of local optima rather than only one.

For comparison purposes, we display below in Table 2.1 the very general differences and similarities of GA and optimisation-applied AIS [Brownlee, 2011, Luke, 2013, Srinivas and Patnaik, 1994]. Implementation details will be exposed in Chapter 5.

Feature	**GA**	**AIS**
Metaphor	Darwinian Theory of Evolution	Mammalian immune system
Individual appellation	Chromosome	Antibody
Performance appellation	Fitness	Affinity or Fitness (CLONALG) / Fitness (AINET)
Encoding	Bit & Real-valued vectors	Bit & Real-valued vectors
Child generation	Crossover	Cloning
Mutation rate	Classically stable, conditioned by mutation rate mp	Varying according to fitness (hypermutation)
Selection	Often stochastic	Best individuals
Expected optima returned	Global optimum	Global and local optima

TABLE 2.1: General comparison between GA and AIS

2.3 Evolving strategies in games

We highlight that, so far, mainly GA has been used on Game Playing problematics. However, due to the close structure of GA and CLONALG (or even opt-AINET), we document in this section GA-based methodologies, from which we could draw inspiration for conceiving an immune-based method.

2.3.1 Analogy Partition for playing the KBC

Vié [2020] has investigated if a GA could learn in the long-term the solution of the (iterated) KBC. The methodology used by the author, consists in considering chromosomes as lists of strategies, and measuring their fitness by confronting each strategy against the target of the list ($p.Avg$). In the case $p < 1$, which interests us in this document, this method succeeded in reaching, in few generations, the Nash Equilibrium and theoretical solution 0, proving a long-term understanding of the game. However, we may note that the framework proposed by the author and the problematic it answers do not quite correspond to ours, as we do not want an agent to play well when $t \to +\infty$, but at every rounds ($t \in [\![1, 10]\!]$). We may highlight though, that the author formulates a measure of a play's performance as an $L2$-Loss, in the fashion of $(x - Target)^2$. Cumulating it over multiple rounds, we can see it as an alternative fitness to optimise (minimise, in this instance) to the final Win Count of the agent. The benefit of using the $L2$-Loss would be that it is real-valued and may have a less steep surface than the integer-valued Win Count.

2.3.2 Memory-based approach to the Prisoner's Dilemma

Another popular game which solutions have been (re)discovered by GAs is the Prisoner's Dilemma (PD). Many variants of this game, with different payoff values, exist, but the simplest description would be the following one [Tucker and Straffin Jr, 1983]: 2 alleged accomplices in a malicious act have been arrested and put in different rooms. Each is given the choice to either confess the mischief or to deny it, but depending on how the other answers, the defendants may have a higher or lower reward or payoff (function of the prison sentence for example). For example if both confess, the sentence would exist but be reduced for both; but if one confesses while the other denies it, the former would have a longer prison sentence and the latter be rewarded. If both deny it, they may be released. In general the game is viewed as a dilemma because the Nash Equilibrium exists and has often a disadvantageous payoff, while taking a risk could be rewarded more positively. The game is often represented by a payoff matrix; for instance Tucker and Straffin Jr [1983] used the following one:

P1 \ P2	**Confess**	**Deny**
Confess	(-1, -1)	(-2, 1)
Deny	(1, -2)	(0, 0)

TABLE 2.2: Payoff matrix of the PD from [Tucker and Straffin Jr, 1983]

An iterative version, like for the KBC, exists where the defendants have to choose to confess or deny during multiple turns and optimise their final payoff: it is the Iterative Prisoner's Dilemma (IPD). Some good strategies have been observed by sociologists, like notably win-stay lose-shift (known as Pavlov) or reciprocal plays (Tit-for-Tat) [Wedekind and Milinski, 1996]. Some tried to emerge such strategies with GAs, which is the case of notably Axelrod [1987] in the first place, whose methodology has been later re-used by Golbeck [2002] to extend the conclusion of the former: using a slightly different terminology from [Tucker and Straffin Jr, 1983] - since actions are called "defect" (equivalent to "confess") and "cooperate" (equivalent to "deny") - but a similar payoff matrix to Table 2.2, her findings state that traits of good playtypes are the ability to cooperate with cooperators, and defend themselves against defectors.

In this game there are only 2 strategies: Cooperate (C) or Defect (D). Golbeck used a representation where an agent fires a strategy in response to the 3 previous memories, which have each 2 values: its choice and the one of the other prisoner. An individual can therefore be seen as a dictionary having $(2 \times 2)^3 = 64$ keys, being the previous possible scenarii, while the value is the response to these memories. For instance, $Dict[CD\ DC\ DD] = C$ means the agent will play C, when in the latest memory it has played C and the opponent D, then played D against C, and finally D against D. Fitness is measured by confronting dictionaries against each other and cumulating the payoff of each scenario outcome.

Alonso et al. [2004] have also investigated the IPD with the same encoding as well, using this time opt-AINET instead of GA. The work concludes that the technique was viable and adaptive to change of strategies given enough time for adaptation against a specific playtype (Always Cooperate, Always Defect, Tit-for-Tat, Pavlov).

These approaches to emerge optimal strategies in function of memory are quite elegant, but may be incompatible to our problem which doesn't have only 2 strategies, but 101.

Eventually, in the memory-1 case - i.e. the agent remembers the most recent average or target and fires a response to it - this method is viable (the dictionary has 101 key-value pairs, in $[\![0, 100]\!]^2$). However, we can be a priori a bit doubtful that the agent could use a single memory to beat level-k players, since there exist many combinations of levels in this game.

2.4 Neuroevolution and adaptation to environment

Neuroevolution is an application of metaheuristics - often GA - in the case where individuals to optimise are neural architectures. We present in this section this paradigm, which can be relevant to our application.

2.4.1 Artificial Neural Networks (ANNs)

ANNs are known as universal approximators of arbitrary precision since [Cybenko, 1989]. This model of Machine Learning mimics neural behaviour, working as a set of hidden layers - constituted of computation units called nodes or neurons - and which are supposed to represent the relationship between an input and an output. When passing an input to the ANN, combinations of weights and their activation by a function - removing the network's linearity - allow to compute an output: it is known as the forward pass. A schematic structure of the regressive case (one output), used in [Connolly, 2020], is represented in Figure 2.3.

The training of the model consists in finding the right connection weights between nodes, and by extension layers. The objective is to minimise a loss value, which measures the error between the prediction of a network and the ground truth. By differentiating the loss in the last layer and using matrix formalism, it is then possible to apply gradient descent to propagate the error from the last layer to the first one, and then update all the weights iteratively: this process is called Backpropagation [Werbos, 1990]. This training technique proves to be very efficient, nevertheless, classical ANNs work as supervised models, i.e. they use a loss function and pairs of input and output to train. When labels are not available (or are difficult to obtain), or even when the loss is not intuitive to implement, it is not possible to adjust to the ground truth using gradient descent. However, in some situations, we may want that an ANN self-adapts without labeled

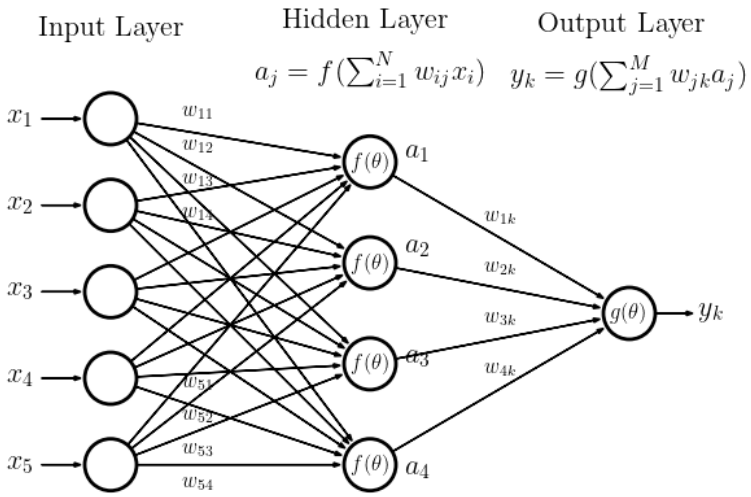


FIGURE 2.3: 1-hidden-layer regressive ANN, from [Connolly, 2020]. Weights w_{ij} are to be found to match outputs and inputs. The activation functions f and g remove the network's linearity and allow it to approximate a wide range of complex functions

outputs and a loss function, which is why Neuroevolution is appealing: by attributing a fitness to its output, the ANN may be able to evaluate itself and evolve towards fitter configurations [Miikkulainen, 2010].

2.4.2 Evolutionary Robotics

Evolutionary Robotics are a branch of Robotics concerned with the implementation of a specific behaviour into robots - often to be autonomous in an environment - by means of genetic processes. For example, Mondada and Floreano [2005] have investigated the possibility to make the Khepera mini-robot, equipped with 2 wheels and 8 proximity sensors (cf. Figure 2.4, from [Mondada and Floreano, 2005]), to learn how to complete an obstacle course with the fewest collisions possible. The controller of the robot was designed as a 1-hidden-layer ANN, having as inputs the values of the 8 proximity sensors, while the output would be the speeds for each wheel (if the left wheel's speed was higher than the right one, then the robot would turn to the right). Then, a fitness function has been designed to allow selection of controller weights to pass to the next generation, reproduce and mutate into progeny weights. It was expressed as a product of 3 sub-fitness functions being respectively a fitness to go straight, avoiding collision to wall, and avoiding to spin on itself. In 150 simulated generations, a robot emerged, able to go through the obstacle course rapidly and without colliding into any wall. Another variant of controller has also been considered, taking into account for example the floor

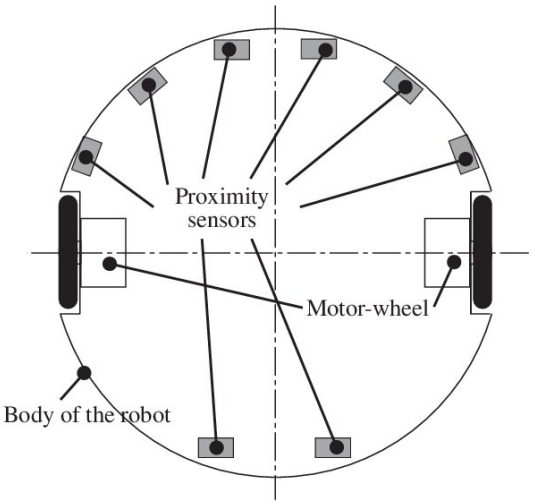


FIGURE 2.4: Khepera robot diagram, from [Mondada and Floreano, 2005].

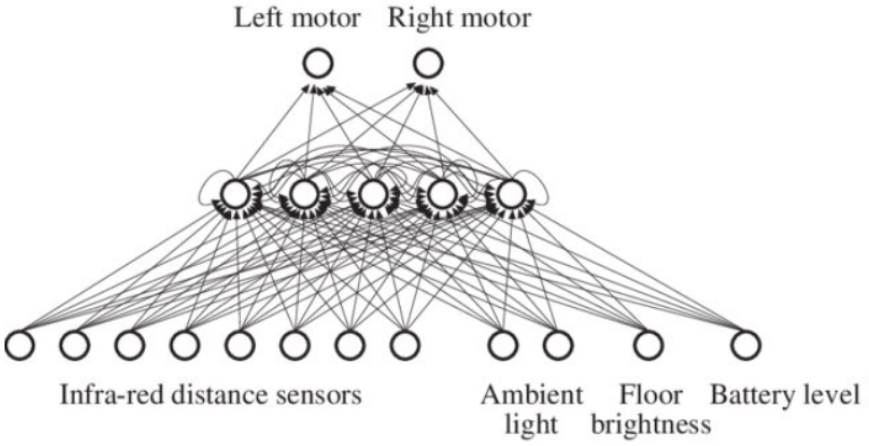


FIGURE 2.5: Extended controller of the Khepera robot, from [Mondada and Floreano, 2005]

brightness to guide trajectory, as seen represented in Figure 2.5 [Mondada and Floreano, 2005]. It is worth mentioning that there is no unicity of the suitable fitness function for this problem, an alternative being for example to sum the sub-functions instead of multiplying them [Pilat and Oppacher, 2005]. Many other applications exist, like generating the behaviour of Unmanned Aerial Vechicles (UAVs) in a swarm [Hauert et al., 2009] or make a NAO robot learn to play soccer [MacAlpine and Stone, 2018].

Regarding our problematic, the main framework of Evolutionary Robotics, or at least the application presented previously, is in fact similar to ours: an agent interacts at each time step with an environment, that to which a fitness may be computed. As for the Khepera robot experiment, we do not know - or do not want to label arduously - the correct output corresponding to each combination of inputs (that would be past results

of the game in the KBC). The fact that ANNs have been investigated successfully to represent robot controllers can be encouraging to do the same with our agent.

However, as noted by Doncieux et al. [2015], using a stochastic method such as GA to train robots prompts to repeat experiments several times to expect correct convergence, which can take a lot of time, which is applicable to our study as well. Now more than before, the increasing power of parallel technologies allows to reduce considerably this computation time though, and make possible some tasks that were not before. We may note that it is often on the condition to parallelise experiments on High-Performance Computing (HPC) clusters. For our application it may be convenient to run calculations on an HPC cluster, even though the task seems to be less intensive than Robotics simulations.

2.4.3 General Game Playing

Another application field of Neuroevolution is General Game Playing, which is concerned with optimising the performance of a player in a particular game, or more. The principle is roughly the same as in Evolutionary Robotics, since an agent confronts in real time an environment, and has a performance against it. The main difference is that the goal is subjectively more recreational, and that there is a priori no physical components which can alter performance (like noise), even though some randomness may be present (e.g. targets that can appear at any location, like in Snake). Among games that were successfully optimised using neuroevolved networks, we can mention for instance Flappy Bird [Brandão et al., 2019], Snake, Pac-Man and Mario Bros [Risi and Togelius, 2014].

The methodology for some is actually quite close to what was done in [Mondada and Floreano, 2005] - with just different outputs and inputs - which supports again the possibility do to so with the KBC. Nevertheless, we may note that all games are not necessarily optimised using a GA, some used other Evolutionary Strategies, which are in fact a larger class of metaheuristics encompassing GA. Another popular evolutionary strategy is for instance CMA-ES (Covariance Matrix Adaptation Evolution Strategy), which drives candidate solutions in preferential directions according to a covariance matrix linked to best individuals [Hansen, 2016]. We may note also that fixed topology networks were used to solve these games, i.e. the user defines the number of layers

and neurons contained of the network to optimise, which falls in fact into the category of Conventional Neuroevolution (CNE), in contrast to NEAT (Neuroevolution of Augmenting Topologies) which evolves the architecture of the network as well.

2.4.4 NEAT

By fixing the topology of a network, one might restrain arbitrarily the range of behaviours that can be reached. The NEAT algorithm has been conceived by Stanley and Miikkulainen [2002] to deal with this limitation. This method doesn't use the conventional encoding consisting in concatenating weights in a vector considered as a chromosome. Instead, it maps a genotype, having two different sets for nodes and connections, matching to a phenotype that is the resulting network, as illustrated in Figure 2.6 [Stanley and Miikkulainen, 2002].

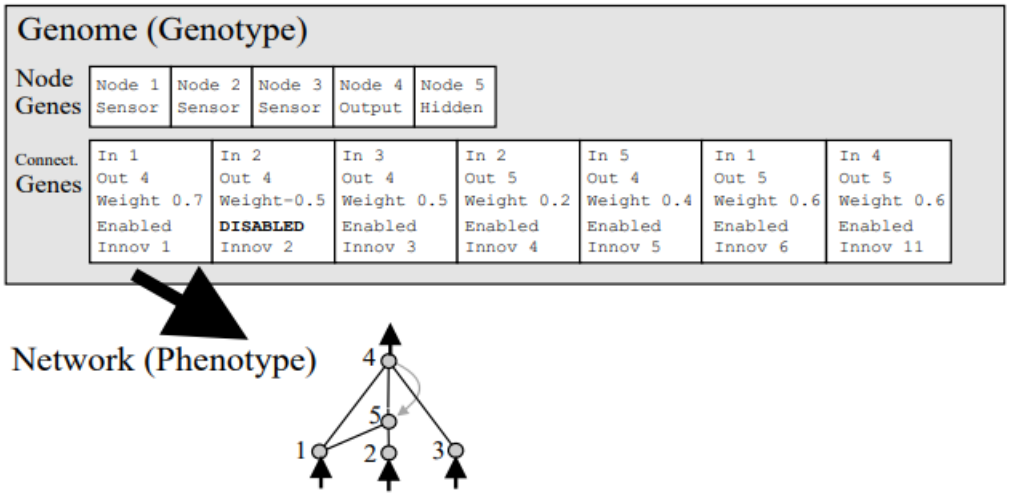


FIGURE 2.6: NEAT encoding of an individual, mapping a genotype to a phenotype (architecture) [Stanley and Miikkulainen, 2002]

The node genes specify the labels of input, output and hidden nodes; while connection genes specify the origin and destination of links, their weights also if they are enabled or disabled. Connection genes are also identified by an innovation number, which is incremented if the connection is new compared to other networks in the population. Crossover is applied between two parents by matching uniquely their connection genes based on their innovation number, and each has a chance to pass a gene to the progeny with probability 1/2 (uniform crossover). As two architectures may not have the same number of connections, excess genes from a parent may be removed if it is less fit.

Multiple types of mutation are considered, like adding a link, adding a node or changing the weight of a connection. Finally, selection is done through speciation, meaning that genotypes are grouped in function of a measure of similarity, and then, less fitted of each group will be discarded. The method displayed outstanding performance when it was introduced, proving to converge in much less generations (and with less evaluations) in the Double Pole balancing with Velocities task than other known methods, including CNE.

More recently, we see that NEAT is still a quality choice for Neuroevolution tasks, being widely adopted in Evolutionary Robotics [Doncieux et al., 2015] or even in General Game Playing [Hausknecht et al., 2014, Risi and Togelius, 2014, Stanley et al., 2005], and being more efficient than CNE in general. It is worth noting that some extensions have also been developed with success [Stanley et al., 2009].

It is essential to note however that no equivalent using an immune-based paradigm has been highlighted yet. It is relevant to wonder if relying only on (hyper)mutation and not crossover can allow to reach similar architectures and performance. Therefore an immune-based alternative is worth investigating.

2.5 Conclusion

Following Literature review, 2 main methods seem appealing to model an agent decision process in the KBC.

The first one is to encode an individual as a dictionary of responses against the previous result (equivalently, the average or the target). This case corresponds to the one of a memory-1 agent, and we can not hope to increase its memory size. Indeed, using 2 memories already increases the number of keys per individual, from 101 to 10201 in the dictionary, which can be computationally intensive to optimise.

The second method would be Neuroevolution. In this paradigm, the agent decision process is designed as an ANN, firing a prediction based on the forward pass of 1 or many memories. A priori, even with 2 memories, it is possible to reduce the number of parameters (weights) to be optimised, in contrast to the dictionary-optimisation paradigm. Weights would be updated by AIS and GA according to a fitness measured either by an

$L2$-Loss or the win count accumulated of the agent. Backpropagation is not suitable to our problem because labelling each output in correspondence to a particular state of the game (including previous averages, and the actual set of player levels) can be tedious; but also because in our framework, opponent players are seen as black boxes and the agent is not supposed to know that they play following rationality levels. Therefore, it would not be fair to train it as if it already "knew" their labeled plays. Conventional Neuroevolution would be preferable to NEAT in our case, which is more advanced and designed to work in conjunction of GA and not AIS. Indeed, regarding our problematic, we want to assess the validity of immune-based methods to optimise the KBC. Moreover, CNE would also enable us to test if a "rudimentary" neural network can be fit for the investigated task. The simplicity of a solution can be indeed more appealing than a complex one. Otherwise, using NEAT, and maybe adapting it to work with a method like CLONALG, remains an very correct alternative that should be worth investigating.

Chapter 3

Requirements Analysis

3.1 Objective of the project

The objective of the research is to assess if AIS can provide a sufficient training framework to build a game playing agent (or predictor) in the KBC, especially in a model where players' decisions are past-dependant. This agent will be implemented in a package, written in Python. Many requirements can be established concerning this project, that we will categorise into Functional and Non-Functional requirements. Additionally we define their priority through the shall (mandatory) / should (optional) pattern.

3.2 Functional Requirements

Functional requirements, which define the features of the project, are the following.

- The package **shall** contain separate agent training and testing components (files or modules)
- An additional component **shall** implement simulated players and make them able to play with the agent
- The training component **shall** propose at least one model of agent, which decision process can be optimised
- The training component **shall** propose at least one immune optimisation technique

- Once trained against simulated players, an agent **shall** be stored (in the form of weights in a text/csv/binary file for example) easily to be re-employed in testing phase
- Training and testing modules **shall** provide options to change the game parameters (Essentially: the number of games to play, the number of players, p and their rationality level k. Eventually: the range of integers and number of rounds in a game)
- The training module **shall** provide options to tune the model's hyperparameters (such as the number of generations, the initial population etc.)
- Training and testing modules **shall** track the results of played games, and enable to display (print or plot) them in a readable way
- The training process **should** be able to be launched on a computing cluster, hosting ideally multi-core CPUs
- The package **should** include a GA-trained implementation, to compare with the AIS one

3.3 Non-Functional Requirements

Non-functional requirements define the quality constraints of the system, and are the following for this project :

- Rounds and games between the agent and simulated players **should** be played fast
- The package **should** be well documented to be as much user-friendly as possible
- The package and programs in general **should** have a structure enabling easy configuration, launching, and be open to extension

Chapter 4

Professional, Legal, Ethical and Social Issues

4.1 Professional and Legal Issues

Codes produced during the project are tested, commented and documented to a high standard. They are moreover written according to coding standards, notably respecting when possible the Open-to-extension/Closed-to-modification principle. Use of any information, already existing codes or tools, useful to carry out the project are referenced, in the codes and (or) the final report. Regarding the legal aspect of the project, no conflict is observed since we only used Python - and its open-source libraries - compatible with the **GNU General Public License**. Data confidentiality is not an issue in this project as no human will be involved.

4.2 Ethical Issues

This research project doesn't come across any ethical issues. No human subject is involved neither in experiments, nor in the evaluation process. No sensitive datasets are used either. Training data are generated through automatised games played between the AIS-based agent and the simulated players.

4.3 Social Issues

This project in its actual form does not come across any social issues as it is a proof of concept proposing a very specific application: beating Level-k-simulated players in the KBC. Eventual extensions of this work, however, may have social implications, notably if the agent is used to play real players and evaluate if they relate to Nagel's Level-k model [Nagel, 1995]. There are not necessarily social issues with that, nonetheless, it is important to state a clear methodology for doing so, to not misinterpret the results of such a study. We can propose for example to design a confidence threshold, linked to the average performance of the agent, for refuting or accepting the hypothesis that players relate to the Level-k model. Ideally, participants would be deceived into believing that they play against only humans, which is an ethical consideration to take into account too.

Chapter 5

Implementation

We expose in this chapter all the implementation details of the project. While many derived techniques are presented (and effectively implemented), like the different selection and crossover types for the GA, not all are actually selected to be tested - due to the computational cost of testing all parameters - as shown later in Chapter 6. More experimentation with the full functionalities of the code[1] is left for further work.

5.1 CLONALG and GA implementation

As highlighted in Chapter 2, CLONALG is the simplest immune-inspired algorithm for optimisation, which can easily be compared to GA. opt-AINET is also worth to investigate, as it has a great ability to detect different optima [De Castro and Timmis, 2002]. However, its additional step of antibody suppression is notably responsible for a high complexity cost, since an affinity (or distance) may be calculated between all pairs of antibody at each iteration [De Castro and Timmis, 2002, de Castro and Von Zuben, 2002]. For this reason and the sake of brevity, we will therefore only compare CLONALG and GA.

5.1.1 CLONALG

We consider a population of P antibodies as the following matrix

[1] The implementation is available at https://github.com/Khaalex1/Keynesian_Beauty_Contest

$$pop = \begin{bmatrix} AB_1 & AB_2 & \dots & \dots & AB_P \end{bmatrix} \in I^{D \times P}$$

With $AB_i = \begin{bmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_D^i \end{bmatrix} \in I^D$; D being the dimension of an antibody and I the range of its coordinates. At each generation, we consider that a fitness function is able to evaluate a population column-wisely, giving the fitness vector

$$F = \begin{bmatrix} f_1 & f_2 & \dots & f_P \end{bmatrix} \in \mathbb{R}_+^{1 \times P}$$

Three main functionalities are required for the implementation of CLONALG [De Castro and Von Zuben, 2002]:

1) **Selection**: Selection is applied at the beginning and the end of a generation. In both cases, it is applied by computing first a fitness vector against a population, and then retrieving a certain number of most fit antibodies.

2) **Cloning**: Cloning is parametrised by the clone rate $cr < 1$, which gives N_c, the number of clones produced from each antibody:

$$N_c = round(cr \times P) \tag{5.1}$$

The cloned population C is therefore pop repeated identically N_c times ($C \in I^{D \times (N_c.P)}$)

3) **Hypermutation**: one very simple mutation, if we consider that each coordinate of an antibody is in a range I, is to substitute them with a value taken randomly (and uniformly) in I. Hypermutation is parametrised by the mutation factor β, which gives the mutation probability mp of the coordinates of C. To put it formally, let x_k^i be the coordinate $k \in [\![1, D]\!]$ of the antibody AB_i of C. Evaluation of AB_i by the fitness function gives a fitness f_i, enabling the definition of mp_i as :

$$mp_i = exp(-\beta.f_i) \tag{5.2}$$

Then, let B and U be random variables following the respective Bernoulli and Uniform distributions $\mathcal{B}(mp_i)$ and $\mathcal{U}(I)$. The mutated x_k^i, which we note $\tilde{x}_k^i$, can be seen as the following random variable :

$$\tilde{x}_k^i = B.U + [1 - B].x_k^i \tag{5.3}$$

It is essential to highlight 2 facts. The first one is that all coordinates k of the same antibody follow the same distribution, enabling to formalise matrix operations. Indeed, once $probas_{mut} = [mp_1, mp_2, ..., mp_{N_c.P}]$ is computed, the mutated population pop_{mut} can be expressed with matrix operations :

$$pop_{mut} = b.u + (1 - b).C \tag{5.4}$$

b realises the Bernoulli distribution in the form of a binary matrix of shape $(D, N_c.P)$, using the parameters in $probas_{mut}$, while u realises the uniform distribution in a matrix of shape $(D, N_c.P)$ as well. The second essential fact is that mp is designed to be high for low fitness antibodies, and vice versa. Therefore it is important to keep β and the fitness of antibodies positive, with the objective of maximising the latter (the closer it is to 0, the closer mp is to 1; and the further it is to 0, the closer mp is to 0). This means that we need to rescale originally negative fitness functions to be positive, for example by adding a constant. Moreover, fitness functions to be minimised need to be rescaled to be maximised, which can be done by changing the sign of the functions (and eventually adding a constant to it). β then needs to be tuned thoughtfully, to generate appropriate mutation probabilities.

Once these main steps have been implemented, CLONALG consists in iterating these latter as presented below in Algorithm 1. This template corresponds roughly to the one proposed by De Castro and Von Zuben [2002], who introduced the method :

Algorithm 1 CLONALG for optimisation

INITIALISATION

pop $\leftarrow$ **random initial population of cardinality** P

AT EACH GENERATION, UNTIL TERMINATION

k most fit antibodies are selected

select_pop $\leftarrow$ **select(pop, k)**

The selected population is cloned N_c times

C $\leftarrow$ **clone(select_pop, N_c)**

Each individual undergoes hypermutation parametrised by β

C $\leftarrow$ **hypermutate(C,** β**)**

The new and old populations are mixed, to preserve good elements regardless of their generation

new_pop $\leftarrow$ **concatenate(pop, C)**

P best elements are selected, reverting the population to its original size

new_pop $\leftarrow$ **select(new_pop, P)**

d less fitted individuals are replaced by randomly generated ones

new_pop $\leftarrow$ **replace(d)**

pop $\leftarrow$ **new_pop**

RETURN

pop, fitness(pop)

5.1.2 GA

We adopt the same formalism as in the previous sub-section 5.1.1, even though the GA individuals are called chromosomes and not antibodies. Moreover, the coordinates of each chromosome are called genes. We still have the real-value encoding:

$$pop = \begin{bmatrix} C_1 & C_2 & \dots & \dots & C_P \end{bmatrix} \in I^{D \times P}$$

With $C_i = \begin{bmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_D^i \end{bmatrix} \in I^D$; and evaluation of *pop* by a fitness function still gives $F = \begin{bmatrix} f_1 & f_2 & \dots & f_P \end{bmatrix} \in \mathbb{R}_+^{1\times P}$.

Three main steps are needed to implement the GA as well [Brownlee, 2011, Luke, 2013]:

1) **Selection**: Selection mimics the natural selection of individuals to be parents for the next generation. In contrast to the clonal selection, it doesn't necessarily consist in selecting most fit individuals. Randomness can be introduced in the mechanism of selection, to reduce risks of premature convergence to a sub-optimal solution. Two popular mechanisms that we implement are the following ones :

   - k**-Tournament Selection**: The Tournament Selection is applied by first choosing randomly k chromosomes in *pop*: $tourney = [C_{t_1}, C_{t_2}, ..., C_{t_k}] \in I^{D\times k}$. A fitness vector is computed for this sub-population: $F_{tourney} = [f_{t_1}, f_{t_2}, ..., f_{t_k}] \in I^{1\times k}$. Then, selection can be done in multiple ways. It could be applied by simply retrieving the chromosome of highest fitness (or lowest one, depending on the problem), but it can also be done by introducing randomness once again, by for example assigning the probability p for the most fit individual to be selected. If not selected, the second one has a probability say $(1-p).p$ to be selected, the third one $(1-p)^2.p$, and so on (the selected rank follows the Geometric Distribution $\mathcal{G}(p)$). Tournament Selection is applied N_{par} times, N_{par} being the number of parents needed for reproduction (crossover).
   - **Roulette Wheel Selection (RWS)**: RWS consists first in evaluating *pop*, and mapping the chromosomes to a fitness vector. Then, this fitness vector in normalised to be a probability vector (which sum is 1). In a minimisation problem, we would have to invert first the fitness values, and then normalise them by the sum of values (a small quantity ϵ may be added to terms to prevent division by zero):

$$probas_{select} = \frac{INV(F)}{SUM(INV(F))} \tag{5.5}$$

Each probability is then mapped to a portion of $[0, 1]$. For example, given $probas_{select} = [0.5, 0.3, 0.2]$, 0.5 would be mapped to $[0, 0.5[$, 0.3 to $[0.5, 0.8[$, and 0.2 to $[0.8, 1[$. If we want to select N_{par} parents, then N_{par} values will be drawn from $\mathcal{U}([0, 1])$, and we will select the chromosomes corresponding to the intervals of each draw. For instance, if we draw $0.63 \in [0.5, 0.8[$, we select the second chromosome (mapped to probability 0.3).

2) **Crossover**: The Crossover operation corresponds to reproduction of parents. Say we want to produce N_c children, then N_c pairs of parents will be chosen randomly for reproduction. A frequently used crossover is known as the 1-point crossover. It consists in choosing randomly a cut (or coordinate) in the chromosome, below which the offspring will have the genome of the first parent, and beyond which they will have the second parent's one. Alternatively, this cut can be considered to be the middle of the chromosome (Middle-point crossover), e.g. the offspring $[A, B, C, D]$ receives $[A, B]$ from parent 1, and $[C, D]$ from parent 2. Crossover is parametrised by a crossover probability (or rate) cp, determining the frequency at which 2 parents reproduce. Otherwise they don't create a progeny, and just pass to the next generation.

3) **Mutation**: The mutation considered on the children population for the GA is a uniform distributed one, exactly the same as the AIS one, mentioned in sub-section 5.1.1.

To remain consistent with population sizes, we choose N_c to be equal to P, so that each generation has the same size. Additional mechanisms implemented are elitism and culling, which allow fitness stability and eventual avoidance of premature convergence. Elitism transfers a certain number of most fit parents to the next generation, which ensures that best individuals are kept even if the children don't surpass the parents. Culling eliminates the weakest individuals and introduce randomly generated ones instead, enabling diversity in the population. The GA then iterates on these functionalities as seen in the pseudo-code 2, which draws mainly inspiration from [Brownlee, 2011, Luke, 2013] :

Algorithm 2 Skeleton for GA (minimiser)

INITIALISATION

Pop ← random initial pop. in domain definition (Card{Pop}=P)

Best_Fitness ← None (or $+\infty$)

Best_Individual ← None

AT EACH GENERATION, UNTIL TERMINATION

S potential parents are selected for crossover

Select_pop ← selection(Pop, S)

Automatically keeps the "pre-selection" best individual

Select_pop ← elitism(Select_pop, Pop, 1)

K children are created by crossing parents (cross. rate = cp)

Children ← crossover(Select_pop, K, cp)

With probability mp, each gene of each child mutates

Children ← mutate(Children, mp)

P best children are selected to be in the next generation

Children ← selection(Children, P)

E elites are reintroduced and $Cull$ less fitted individuals are replaced by randomly generated ones

Children ← elitism(Children, Pop, E)

Children ← culling(Children, Cull)

The children are ready to be parents in the next generation

Pop ← Children

Mapping each individual to their fitness in a list (or dictionary)

F ← fitness(Pop)

The best fitness is extracted

Best_Fitness ← MIN(F)

The best individual is extracted

Best_Individual ← Pop[ARGMIN(F)]

RETURN

Best_Individual, Best_Fitness

5.2 Dictionary-optimiser agent: Dict-opt

Following the model of response to memory formulated by Axelrod [1987] and then Golbeck [2002] in the Iterated Prisoner Dilemma, we can consider a first single-memory agent, which decision process can be represented as a dictionary: each key represents the previous average of the game, while the corresponding value would be the response to this memory. For instance,

$$config = \{..., \ 50 : 47, \ 51 : 49, \ ...\} \tag{5.6}$$

means that the agent plays 47 when $Avg[t-1] = 50$ and 49 when $Avg[t-1] = 51$. In fact, the data structure of this agent could be simplified to a list or array, where the index implicitly indicates the value of $Avg[t-1]$, e.g. the value at index 2 is the response to $Avg[t-1] = 2$. Therefore, a configuration of agent is represented a priori as a list of length 101, where each element is an integer in the range of choices in the KBC: $[\![0, 100]\!]$.

Such a list represents an individual which fitness can be optimised by both CLONALG and GA, but at the condition of initialising it in $[\![0, 100]\!]^{101}$ and making the mutation still uniform, but on $[\![0, 100]\!]$ as well.

Since there is no memory in the first round yet, we make Dict-opt play 50 by default at this stage.

As noted before, considering such an agent with more memories is computationally unrealistic: in the memory-2 case, an agent configuration is naively (without prior knowledge of the game dynamics) represented as a dictionary of $101 \times 101 = 10201$ key-value pairs, which is already extremely high for a single individual.

5.3 Neuroevolved Regressive Networks: RegNet1 and RegNet2

In order to develop the agent's decision process as a regressive neural network, which we abbreviate as **RegNet**, we must reflect on its input and output. The output is simply a decision in the KBC, so an integer in $[\![0, 100]\!]$. Therefore, a suitable output activation

is :

$$out\_act(W, B, Z) = \lfloor 100 \ . \ \sigma\big(W \times Z + B\big) \rfloor \tag{5.7}$$

With W, B and Z respectively the last-hidden-layer weight matrix, bias and input vectors. The matrix product $(W \times Z)$ and B are uni-dimensional ($\in \mathbb{R}^{1\times1} \equiv \mathbb{R}^1 \equiv \mathbb{R}$), making the output uni-dimensional as well. The non-differentiability of this integer-valued activation function is not compatible with the usual gradient-descent-based training, but is acceptable in our metaheuristic-based paradigm. Concerning the input, we know that the only information available in this game are the previous round average and target. These can be seen as equivalent possibilities of input since they always differ by the same factor. Intuitively, the memory of the 2 previous results can be sufficient to determine a pattern between the previous rounds and to launch a response. However, it can be interesting to test whether the only memory of the last round can be enough to fire an appropriate response or not.

We have to note that in the first round(s) where memory has not been constituted yet, we force the agent to overrule the forward pass, and play by default 50. Increasing the memory size is counter-productive in terms of possible victories because a game lasts only 10 rounds. Indeed, if we want to make the agent decide in response to 3 previous memories, it will have to play by default during 3 rounds, so we can expect it to win at most 7/10 games. It can actually still win some of the idle rounds, but likely only against very low-level crowds.

Therefore, we propose 2 arbitrary and fixed architectures - represented in Figures 5.1 and 5.2 - which weights are to evolve, with:

- **Input layer (IL)**: 1 or 2 features, representing the previous round(s) average result(s). They are always rounded to the nearest integer, which mimics the human computational capacity
- **Hidden Layer 1 (HL1)**: 10 neurons, activated by $reLu$
- **Hidden Layer 2 (HL2)**: 10 neurons, activated by $reLu$
- **Output Layer (OL)**: 1 neuron activated by out_act, providing the prediction for the next round target

The *reLu* activation function is arbitrary and fixed. This choice is based on the empirical findings that reLu often obtains better results than sigmoid or tanh in classical Backpropagation training [Krizhevsky et al., 2012], even though it is not the training method here.

To neuro-evolve such architectures, we consider that an agent or individual is represented by a vector containing consecutively all their flattened weights and biases. Thus, they have a dimension of 141 or 151 [2]. These architectures were chosen in order to limit these individual sizes and not burden the AIS (and GA) efficiency. Each weight can be bounded - at the initialisation and during the whole training - on the interval $[-1, 1]$. This choice allows to restrict the search space, while - combined with the action of reLu - still giving magnitude to the pre-activated output (cf forward-pass Equation 5.8 below). It is indeed important for the predictions to cover the range $[\![0, 100]\!]$[3].

Only the forward pass is needed to compute the decision of the agent, while it will be evaluated by a fitness function - defined in the following section (5.4) - to select the most fit ones for crossover, mutation or (and) transferring to the next generation. Let $i \in [\![1, L]\!]$ be the index of a layer (1 being the input layer and L the output one), the forward pass computes the output Z of each layer iteratively as [Huang et al., 2004]:

$$\forall i \in [\![2, L]\!], \begin{cases} Z^i &= W^i.O^{i-1} + B^i \\ O^i &= activation^i(Z^i) \end{cases} \tag{5.8}$$

With $O^1 = \begin{bmatrix} Avg[t-2] \\ Avg[t-1] \end{bmatrix}$ (or $= \Big[Avg[t-1]\Big]$) and $O^L = \Big[prediction[t]\Big]$. This allows to define the function `forward`, with the following signature :

`forward(weights, memory)` :

- **parameter** `weights`: list or array of floats, contains the flattened weights and biases representing an agent
- **parameter** `memory`: list or array of integers, the input vector for the RegNet containing the previous average(s) (O^1)

[2] $Number\ of\ parameters = (IL\ size + 1) \times HL1\ size + (HL1\ size + 1) \times HL2\ size + (HL2\ size + 1) \times OL\ size$

[3] Actually 100 cannot be reached, which isn't really a problem though since it is always the worst play, according to iterated dominance

- **RETURN** `prediction`: integer, the output of the forward pass ($prediction[t]$) which is the response of the agent (`weights`) to `memory`. `weights` is first hashed and reshaped to fit the dimensions of the W^i and B^i before proceeding to the matrix operations (cf. eq. 5.8)

We must note that the proposed architectures somewhat abusively assume that they can produce acceptable results if the weights are correctly found, which are not guaranteed to exist. A proper methodology would be to test many architectures, or even using an algorithm close to NEAT [Stanley and Miikkulainen, 2002] to evolve the architectures as well as the weights during the training. However, we will test this simple assumption first - which enables to limit the complexity of the hyperparameter investigation - and eventually change the architecture in case of "bad" performance.

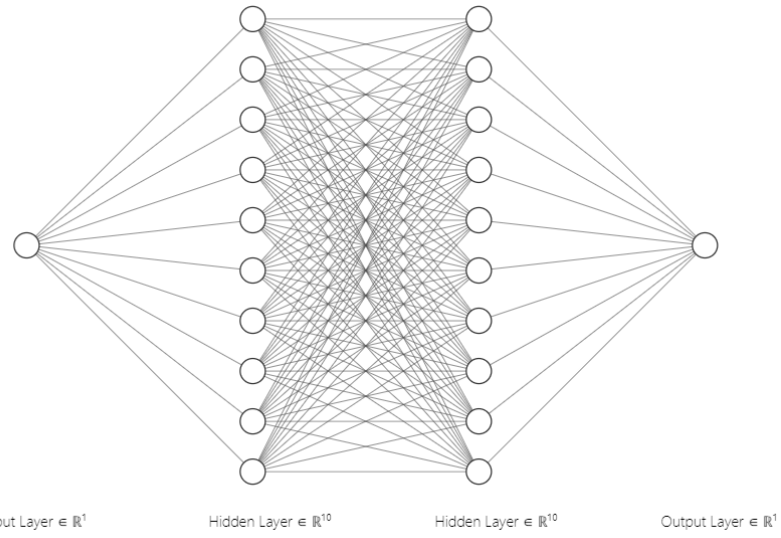


FIGURE 5.1: RegNet1 (memory-1 network), drawn using [Lenail, 2018]. The input feature is the previous round average result

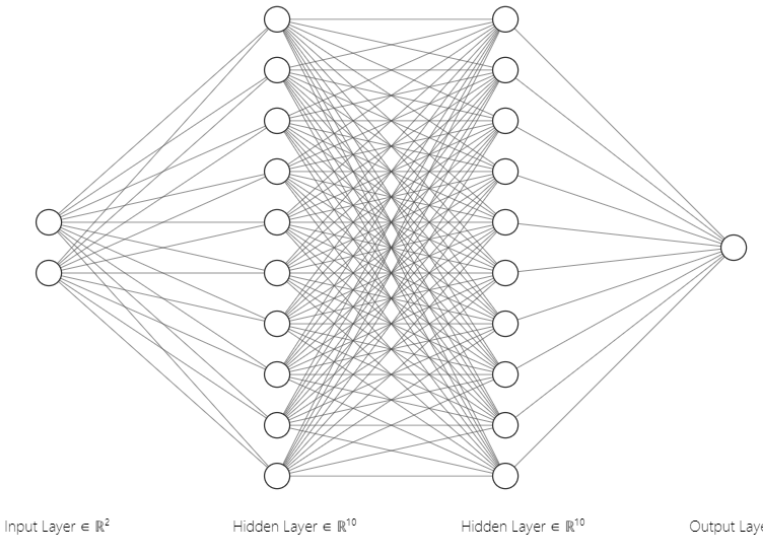


FIGURE 5.2: RegNet2 (memory-2 network), drawn using [Lenail, 2018]. The input features are the 2 previous average results

5.4 Fitness value for the agent

5.4.1 Crowds of rational players

Following Nagel's findings [Nagel, 1995], simulated players can be modelled having a static rationality in $\{0, 1, 2, 3\}$. As noted in Chapter 2, this model is purely theoretic and doesn't leave place to dynamic change of rationality (for example, adjusting strategies in function of losses in the game). Even though this model matches Nagel's results, gaps were observed and it is conceivable that some players - even if not the majority - play randomly, are highly rational by exceeding the level-3, or tend to change their rationality following the course of the game. However, it is not particularly a problem for the sake of this study, which considers first that this model provides a dynamic benchmark [4] for neuroevolved regression and decision taking.

We choose to have a crowd of 5 players, so the agent plays against 4 simulated players. This choice is purely arbitrary, but in Section 6.7, we will test if an agent trained against 4 players can still extrapolate plays against larger groups.

[4] Rationality levels (or if we want, patterns of play) are fixed, but players still change their play after each round

Since an agent always plays against 4 players, who can each have any rationality in $\{0, 1, 2, 3\}$, some combinatorial calculations allow to compute the number of possible groups. The order of group permutations doesn't matter e.g. playing against the rationality group $\{0, 1, 2, 3\}$ is the same as playing against $\{1, 3, 2, 0\}$. Moreover, each item in $\{0, 1, 2, 3\}$ can be repeated infinitely e.g. the groups $\{0, 0, 0, 0\}$ and $\{2, 2, 2, 2\}$ are possible. Consequently, here we consider combinations with repetitions, calculated by :

$$\binom{n+r-1}{r} \tag{5.9}$$

With r the number of draws, from a set of n different objects which can be repeated infinitely. Therefore, there are $\binom{4+4-1}{4} = \mathbf{35}$ different rationality combinations, which represent the different groups the agent can play against.

The level-k model states that a player of rationality k plays $P[t] = Avg[t] \times p^k$. To design simulated players, we choose like in Section 5.3 to round $Avg[t]$ to the nearest integer, and then $P[t]$ as well. We also modify slightly the model by adding a Gaussian noise $\mathcal{N}(0, 0.25)$ to $P[t]$ before rounding it, which makes the simulated players stochastic. Thus, for the purposes of notation:

$$\begin{gathered} \forall t \in [\![2, 10]\!], \\ P[t] = Round\big(Avg[t-1] \times p^k + \mathcal{N}(0, 0.25)\big) \end{gathered} \tag{5.10}$$

This choice of the variance allows to keep the majority of choices (99.7%) within approximately 3 standard deviations ($\sigma = \sqrt{0.25} = 0.5$) of the deterministic play, which is reasonable to introduce stochasticity while not deviating too much from the Level-k model. In the first round, the players are supposed to play $P[1] = Round\big(50 \times p^k\big)$. We widen the range of possibilities like previously:

$$P[1] = Round\big(50 \times p^k + \mathcal{N}(0, 9)\big) \tag{5.11}$$

5.4.2 Game-simulation functions

The three main components of our study are the simulated player, the agent and the crowd of players. Either choosing to apply an Objected Oriented Programming (OOP) paradigm or just designing functions are equivalent. OOP allows to give code clarity

and leaves place to code extension, whereas just calling simply functions can be more optimised for dealing with a lot of calculations. Here, we choose to design only functions to make the code more compact and optimise function calls. For the sake of brevity, we detail only the signatures of these:

1) `player_decision(k, p, prev_avg)`:

- **parameter** `k`: integer, the rationality level of the player
- **parameter** `p`: float, the parameter p of the KBC (by default 2/3)
- **parameter** `prev_avg`: integer, the rounded (to the nearest integer) average result of the previous game. By default -1 (first round)
- **RETURN** `play`: integer, the play according to the level-k model, with an additional Gaussian noise: `play = round(prev_avg` $\times$ `p`$^{\texttt{k}} + \mathcal{N}(0, 0.25)$`)`. Random play if `prev_avg` $\notin [\![0, 100]\!]$ (must happen in the first round)

2) `Dict_opt_decision(agent_config, list_prev_avgs)`:

- **parameter** `agent_config`: list or array of integers, corresponding to Dict-opt responses to their index
- **parameter** `list_prev_avgs`: list or array of integers, represents the previous average(s) rounded to the nearest integer
- **RETURN** `play`: integer, the target predicted by the agent for the next round - in function of the latest result - as `agent_config[list_prev_avgs[-1]]`. When memory doesn't exist yet (empty or incomplete `list_prev_avgs`), the default play is 50

3) `RegNet_decision(agent_config, list_prev_avgs)`:

- **parameter** `agent_config`: list or array of floats, corresponding to the weights and biases for the RegNet
- **parameter** `list_prev_avgs`: list or array of integers, represents the previous average(s) rounded to the nearest integer
- **RETURN** `play`: integer, the target predicted by the agent for the next round as `forward(agent_config, list_prev_avgs)`. When memory doesn't exist yet (empty or incomplete `list_prev_avgs`), the default play is 50

Note: 2 versions can be implemented (for RegNet1 and RegNet2), or embedding both in a single function is also valid

4) `crowd_iterate_play(list_k, agent, agent_config, p, nb_rounds)`:

- **parameter** `list_k`: list of integers, which correspond to the rationality levels of the 4 simulated players
- **parameter** `agent`: function representing an agent, which can fire a decision in response to memory (either `Dict_opt_decision` or a `RegNet_decision`)
- **parameter** `agent_config`: list or array of floats, corresponding to the weights and biases for the RegNet
- **parameter** `p`: float, the parameter p of the KBC (by default 2/3)
- **parameter** `nb_rounds`: integer, the number of rounds simulated with the 4 players and the agent (by default 10)
- **RETURN**
  - `wc_list`: list of integers, corresponding to the win counts of each player including the agent
  - `performance`: float, the performance of the agent at the end of all the rounds played against `list_k` (either the game loss - eventually rescaled - or the win count as seen below in sub-section 5.4.3)

Note 1: The win count of one player is incremented if its choice is the closest to the target. It is possible that many players win simultaneously if they play the same integer

Note 2: This function iterates using `player_decision` and `Dict_opt_decision` or `RegNet_decision`

5.4.3 Fitness definitions and rescaling

We propose an iterative method to define a fitness value for the agent. First it is important to keep in mind that we want to maximise the number of wins the agent can get by playing the game. So the win count can be considered as a suitable fitness function. However as it is a flat-surface (integer-valued) function, its optimisation may be difficult depending on the hyperparameter investigation. It encourages to find other

alternatives that would implicitly optimise the win count of the agent. A squared loss can be defined in this game, which is an idea inspired by Vié [2020]. Indeed, at the end of each round, it is possible to measure the difference between the agent's play and the actual target, in the fashion of :

$$Loss[t] = (Agent[t] - p.Avg[t])^2 \tag{5.12}$$

As we want the agent to be as close as possible to the target at each round, we would like to minimise this loss. When playing against a particular crowd, we estimate the performance of the agent by cumulating its loss over all the active rounds, since we have to keep in mind that it plays by default 50 in the first 1 or 2 rounds ("memory rounds"). Finally, as there are 35 crowds of players possible, we can aggregate this loss when the agent plays against the latter, enabling to compute the total fitness, abbreviated as $L2$:

$$L2 = \sum_{C=1}^{35} \sum_{\substack{t= \\ mem.size+1}}^{10} (Agent[t] - p.Avg_C[t])^2 \tag{5.13}$$

The C index which is used on Avg_C, implies that this value is obtained when the agent plays against the crowd C.

This fitness is very suitable for a minimisation carried by the GA, which can work without rescaling as either a minimiser by taking $ARGMIN(F)$, or a maximiser with $ARGMAX$ (cf. Algorithm 2). However, for the AIS, as noted in sub-section 5.1.1, it is essential to turn the problem into a positive maximisation one. Therefore, we must transform the loss into a positive quantity to maximise. It can be done by negating the loss and then adding a constant to make it positive. This constant can be taken as the squared maximum difference possible between the agent and the crowd, which is 100. Thus, the transformed loss[5] and L2-fitness are :

$$\begin{cases} Loss_{new}[t] = 100^2 - Loss[t] = 10000 - (Agent[t] - p.Avg[t])^2 \\ L2_{new} = \sum_{C=1}^{35} \sum_t Loss_{new,C}[t] \end{cases} \tag{5.14}$$

Minimising $L2$ and maximising $L2_{new}$ are strictly equivalent, so from here we will use mainly the terminology of "Loss minimisation" to make reading easier.

[5] Rigorously it is not a loss anymore since it is to be maximised.

In the same fashion, the Win-Count fitness is defined as :

$$WC = \sum_{C=1}^{35} \sum_{\substack{t= \\ mem.size+1}}^{10} Win_C[t] \tag{5.15}$$

With $Win_C[t] = 1$ if the agent wins against the crowd C at t, or else $Win_C[t] = 0$. We will also attempt to optimise it with both the GA and AIS.

Ultimately, we detail the algorithm allowing to compute the fitness ($L2$ or WC) of an agent configuration at each generation as follows:

Algorithm 3 Fitness computation: $fitness(agent\_config)$

Initialisation

$agent\_config \leftarrow$ configuration list of an agent (Dict-opt/RegNet1/RegNet2)

$CROWDS \leftarrow$ List of 35 sub-lists, the rationality combinations

\# *e.g.* $[0, 2, 3, 2]$ *is a rationality combination*

$fitness \leftarrow 0$

For each crowd in CROWDS

wc_list, performance $\leftarrow$ *crowd_iterate_play(crowd, agent_config, 2/3, 10)*

$fitness \leftarrow fitness + performance$

Return

$fitness$

Chapter 6

Results

6.1 Experimental Protocol

6.1.1 Evaluation protocol

6 optimisation frameworks will be tested: 2 different fitness functions (WC and $L2$) for 3 models of agents: Dict-opt, RegNet1 and RegNet2. Each will be optimised by both CLONALG and GA. Therefore, there is a total of 12 experiments. Concerning hyperparameter investigation, because a grid search can be time-consuming (and computationally intensive), we propose to test simplistic sets that we present below. A reasonable training time is considered in our experiments, which we fix at 500 generations. The mapping between hyperparameter sets and converged performance obtained at the end of an experiment can be presented as the following Table (6.1). The superset of all hyperparameter sets is noted HP and has a cardinality of h.

Hyperparam. set	Best Avg. Win Count	Avg. $L2$-Loss	Agent config.
HP_1	WC_1	$L2_1$	$[x_1^1, x_2^1, ...]$
HP_2	WC_2	$L2_2$	$[x_1^2, x_2^2, ...]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
HP_h	WC_h	$L2_h$	$[x_1^h, x_2^h, ...]$

TABLE 6.1: Mapping obtained at the end of an experiment. There is a total of 12 such tables ($Nb\ experiments = |fitness\ functions| \times |agents| \times |metaheuristics|$)

The agent's performance is measured against a noisy environment (all possible crowds). There is no guarantee that doing well one time against the environment means that it will always do well against it, notably due to the noise introduced in the Level-k model. As a consequence, in our evaluation protocol we consider the final performance of a converged agent as its average Win Count and $L2$-Loss when facing the environment $n = 10$ times (these are $Best\ Avg.\ Win\ Count$ and $Avg.\ L2-Loss$ in Table 6.1). Still, it is important to take into account that convergence may be unstable due to the stochastic nature of metaheuristics. We verify convergence consistency by selecting the final individual as the best agent of median fitness, when running the training 3 times. For instance, say that given a hyperparameter set HP_i we ran 3 training sessions, which gave each a best agent with the following Win Count averages: $WC_{List} = [195.8,\ 193.2,\ 207.5]$. The "Best Avg. Win Count" selected will be $MEDIAN\Big(WC_{List}\Big) = 195.8$, and the best agent, the one that corresponds to it.

We present below the pseudo-code 4 for evaluating the converged performance of each hyperparameter set. The most efficient hyperparameter set for an agent will be considered as the one which training produces the highest Win Count average, which can be seen as $ARGMAX\Big\{WC_{DICT}\Big\}$[1].

Algorithm 4 Evaluation protocol given a framework

For each hyperparameter set HP_i

Run training **3** times, with pop. of P individuals

Retrieve the fitter agent for each of the 3 converged pop.

Selected agent $\leftarrow$ agent of median fitness among the three latter

Compute WC_{Avg} and $L2_{Avg}$ of Selected agent VS. environment **10 times**

 # WC_{DICT} and $L2_{DICT}$ are dictionaries

$WC_{DICT}[HP_i] \leftarrow WC_{Avg}$

$L2_{DICT}[HP_i] \leftarrow L2_{Avg}$

Return

WC_{DICT}, $L2_{DICT}$

In a posterior session, we study the repeatability of results when training each agent 5 times with the best parameters found. We then compute the average Win Count (or $L2$

[1] In a more loose formalism, $ARGMAX\Big\{Best\ Avg.\ Win\ Count\Big\}$, by referring to Table 6.1

value), the standard deviation, maximum and minimum values of the converged agent.

6.1.2 Set of parameters tested for the GA

We choose to test $pop.\ size \in \{10, 20, 40\}$. A maximum test size of 40 can seem a little low but already represents a computational cost of $350 \times 40 = 14000$ rounds (without applying yet the genetic operations). Then, we define the selection rate *select. rate* as the ratio between the selected population size, and the original population size :

$$select.\ size = \lceil select.\ rate \times pop.\ size \rceil \tag{6.1}$$

We will test only the Roulette Wheel Selection to keep a reasonable number of parameters to test. The tested Crossovers will be the Uniform and Middle-point ones. In general, the choice of the crossover and mutation rates depends on the optimisation problem. However, some general ranges have been proposed to ensure convergence, for example by taking a crossover rate in $[80\%, 95\%]$ and a low mutation rate around 10% [Hassanat et al., 2019]. We push this logic to the extreme by taking a crossover rate of 1, which means that crossover is always exploited. It is a priori still viable, since we can rely on elitism to keep best progenitors throughout generations. The mutation is uniformly distributed, and we investigate the probability of mutation $mut.\ rate \in \{0.05, 0.15\}$. Finally, defining the elite and replacing rates as

$$Nb.\ elites = \lceil elite\ rate \times pop.\ size \rceil \tag{6.2}$$

$$replaced\ pop. = \lceil rep.\ rate \times pop.\ size \rceil \tag{6.3}$$

we investigate $elite\ rate,\ rep.\ rate \in \{0.05, 0.15\}$. The tested parameters are summarised below, in Table 6.2:

pop. size	*select. rate*	*crossover*	*mut. rate*	*elite & rep. rate*
$\{10, 20, 40\}$	$\{0.25, 0.5\}$	Uniform (U.) & Middle-point (M.-P.)	$\{0.05, 0.15\}$	$\{0.05, 0.15\}$

TABLE 6.2: Hyperparameters investigated for the GA

We systematically pair up the highest mutation rate with the lowest elite rate, and vice versa, to find a trade-off between computation time and search. In total, this represents 24 combinations of hyperparameters to test for the GA.

6.1.3 Set of parameters tested for the CLONALG

We keep some of the sets used for GA: *pop. size* $\in \{10, 20, 40\}$, *select.rate* $\in \{0.25, 0.5\}$ and *rep. rate* $\in \{0.05, 0.15\}$. The clone rates *cr* will be taken in $\{0.25, 0.5\}$.

For the hypermutation, which is - of the same type as in the GA - a substitution uniformly distributed on $[-1, 1]$ (or $[\![0, 100]\!]$ for Dict-opt), we define first the magnitude *mag* of the fitness function as the power of 10 of its maximum value (which we happen to know[2]). The idea behind choosing the mutation factor β, is that we need it to counter the fitness magnitude, and make $e^{-\beta.fitness}$ an appropriate probability for mutation realisation. Thus, we choose to test the product $\beta \times mag$ in the set $\{0.5, 1, 2\}$, which allows the mutation probability to decrease roughly to the lowest bounds $\{0.22, 0.05, 0.002\}$[3]. As in the GA, we investigate *rep. rate* $\in \{0.05, 0.15\}$. The tested parameters are summarised below, in Table 6.3.

pop. size	*select. rate*	*clone rate*	$\beta \times mag$	*rep. rate*
$\{10, 20, 40\}$	$\{0.25, 0.5\}$	$\{0.25, 0.5\}$	$\{0.5, 1, 2\}$	$\{0.05, 0.15\}$

TABLE 6.3: Hyperparameters investigated for CLONALG

We each time pair up the highest clone rate 0.5 with the lowest replace rate 0.05 and vice versa. The motivation is mainly to find a trade-off between computation time and effective grid search. It gives us a total of 36 combinations of hyperparameters to test for CLONALG.

6.2 Results when maximising $fitness = Win\ Count\ (WC)$

When maximising the Win Count (WC), we select the best agents obtained by grid search as the ones with the highest Win Count average. The performance of such

[2] for the $AIS\_L2\_fitness$ $mag = 10^6$, and for the Win Count $mag = 10^2$
[3] $min.\ mut.\ prob. \approx exp(-\beta \times 3 \times mag)$

agents is summarised below in Table 6.4, and the hyperparameters used to find them are presented in Table 6.5. Even though the $L2$-Loss is not optimised, it can still be measured against the environment, and we display it along with the Win Count.

Here, the $L2$-Loss is normalised by a factor equal to the number of rounds measured (**L2 ← L2/Active Rounds**) [4], which is 315 (9×35) for memory-1 agents and 280 (8×35) for memory-2 ones.

In the same fashion, we define the Absolute Win Rate as **Avg. WC/350** and the Relative Win Rate as **Avg. WC/Active Rounds**.

	Dict-opt		RegNet1		RegNet2	
Training method	CLONALG	GA	CLONALG	GA	CLONALG	GA
Best Avg. Win Count	252.6	123.0	265.6	263.3	245.5	239.3
Absolute Win Rate	72%	35%	76%	75%	70%	68%
Relative Win Rate	80%	39%	84%	84%	88%	85%
Corresponding Avg. L2	19.37	1246.60	2.07	2.18	1.98	2.94

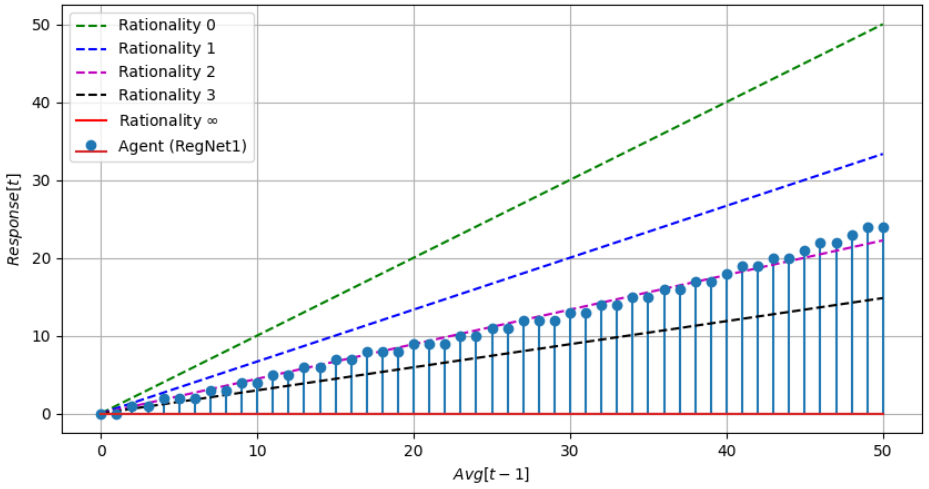
TABLE 6.4: Performance of best agent configurations obtained by grid search in 500 generations ($fitness = WC$)

	Dict-opt		RegNet1		RegNet2	
Training method	CLONALG	GA	CLONALG	GA	CLONALG	GA
pop. size	40	40	20	40	40	10
select. rate	0.5	0.5	0.25	0.25	0.5	0.25
clone rate	0.25	**N.A.**	0.5	**N.A.**	0.25	**N.A.**
$\beta \times mag$	2	**N.A.**	2	**N.A.**	2	**N.A.**
crossover	**N.A.**	U.	**N.A.**	U.	**N.A.**	M.-P.
mut. rate	**N.A.**	0.05	**N.A.**	0.05	**N.A.**	0.05
Elite & (or) rep. rate	0.05	0.15	0.15	0.15	0.05	0.15

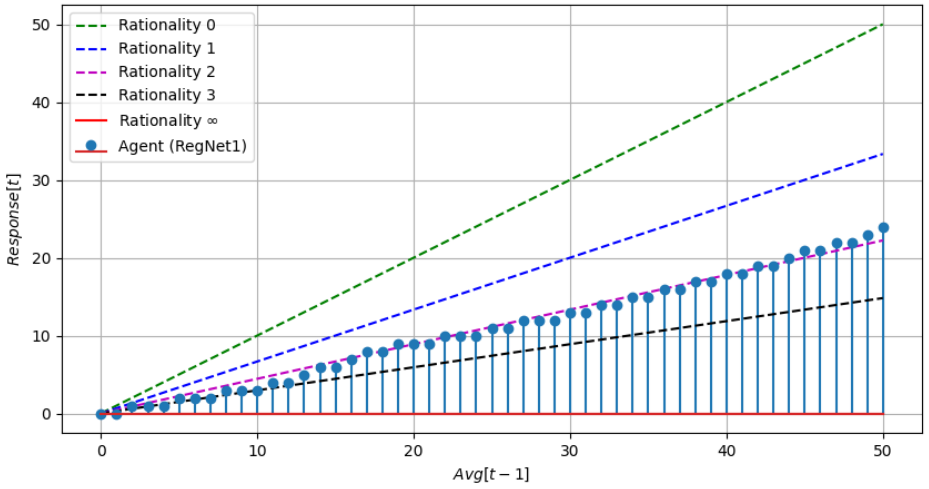
TABLE 6.5: Training hyperparameters used to obtain best configurations in 500 generations ($fitness = WC$)

[4] The Win Count is measured and updated from round 1, but the Loss is measured only from $t = memory\ size + 1$ (cf Section 5.4)

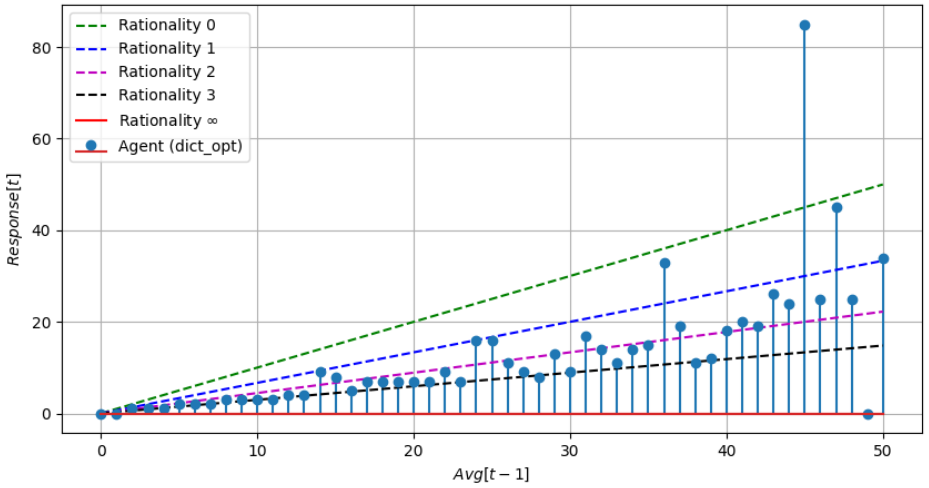
We observe surprisingly exceptional performances from memory-1 agents, in particular the CLONALG-trained Dict-opt and RegNet1 (regardless of the metaheuristic used). They respectively have an absolute win rate of 72% (252.6/350) and $75-76\%$ (263.3/350 and 265.6/350). While the RegNet1 loss averages are low, it is significantly higher for the CLONALG-trained Dict-opt. This is not entirely paradoxical, and translates the fact that this agent wins most of the time, even though it is sometimes far from the target. More insight can be obtained from the response graphs below (6.1) of these agents, obtained by applying the agent functions on $[\![0, 50]\!]$. The CLONALG and GA-trained RegNet1 have a very similar response, having a smooth and increasing evolution. Comparison with Level-k playtypes shows that both RegNets generally play along the level-2 curve - sometimes one integer lower - and eventually shift to the level-3 approximately from $Avg[t-1] = 10$ to 0. In contrast, the Dict-opt response is not monotonous. High peaks above the level-2 may be the cause of high loss values, while plays below $Avg[t-1] = 23$, often regular and close to the level-3, may still allow to win consistently. One similarity between the 3 agents we can note though, is their ability to shift to the Nash Equilibrium 0 near $Avg[t-1] = 0$, which is a sign of good behaviour.



(A) Response of CLONALG-trained RegNet1



(B) Response of GA-trained RegNet1



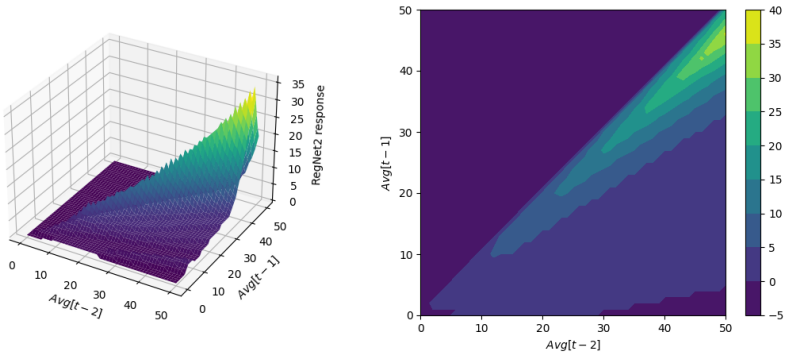
(C) Response of CLONALG-trained Dict-opt

FIGURE 6.1: Response to memory of best agents (*objective = Win Count*)

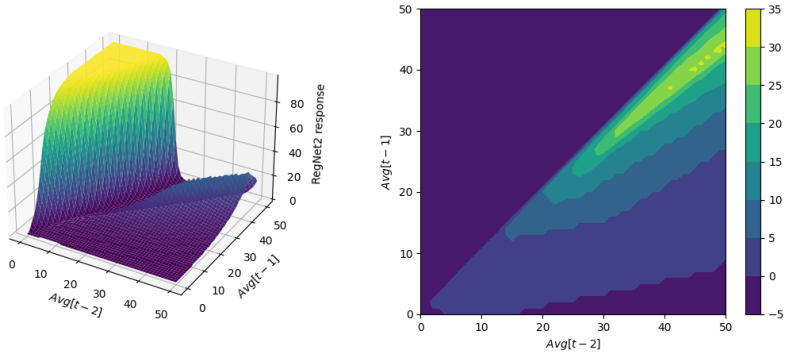
The RegNet2 has a lower performance globally, which can seem paradoxical since it should be more robust by using 2 memories instead of 1. However, these slightly lower results are also explained by the drawback of using 2 memories: indeed, the RegNet2 agents sacrifice 2 turns where they play by default (and are likely to lose) before constituting their first memories. So the relative win rate for the CLONALG-trained RegNet2 is in fact 88% (245.5/280), while the RegNet1 has a relative rate of 84% (265.6/315). This means that RegNet2 exploits better the active rounds than RegNet1, but its performance is heavily burdened by the sacrifice of the $2 \times 35 = 70$ idle turns. We can see in the response - displayed in 6.2 - of both CLONALG and GA-trained Nets that they reproduce a smooth and increasing pattern projected on the plane $\big(Avg[t-2] = 50\big)$; however, the surface still depends on $Avg[t-2]$. The surfaces of both RegNet2 seem different, but only the points verifying $Avg[t-1] < Avg[t-2]$ are actually important and realistic (plays always tend to decrease). The contour plots of these responses, removing the region $(Avg[t-1] > Avg[t-2])$, confirms that both have roughly the same response, which has the convenient characteristic to decrease plays the further $Avg[t-2]$ is from $Avg[t-1]$ - demonstrating adaptation - and to shift to 0 near $\big(0, 0\big)$.

On the other hand the GA-trained Dict-opt has a rather poor performance. Its convergence is not complete, as shown in its erratic response, in 6.3.

Finally, concerning the found hyperparameters, it is difficult to extract generalities from them. Though, we can raise that high population sizes are often preferred, the mutation rate of 0.05 has always been optimal for the GA, and the product $\beta \times mag = 2$ was shared among best individuals.



(A) 3-D response and contour plot of CLONALG-trained RegNet2. The section verifying $(Avg[t-1] > Avg[t-2])$ has been removed in the contour plot



(B) 3-D response and contour plot of GA-trained RegNet2. The section verifying $(Avg[t-1] > Avg[t-2])$ has been removed in the contour plot

FIGURE 6.2: Response surfaces of RegNet2 ($objective = WC$)

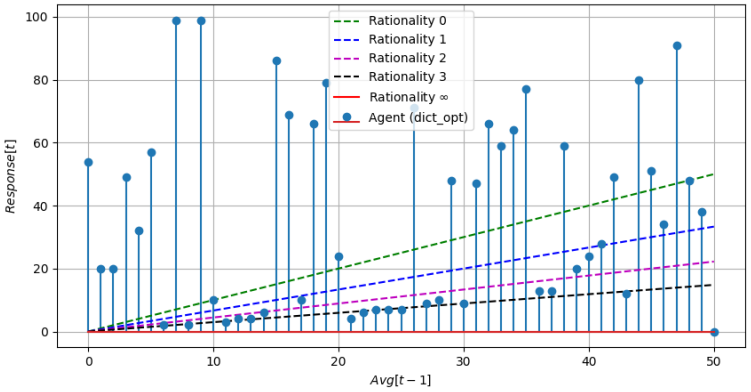


FIGURE 6.3: Response of GA-trained Dict-opt ($objective = WC$)

6.3 Results when minimising $fitness = L2\ Loss\ (L2)$

When minimising the $L2$-Loss ($L2$), we still select the best agents obtained by grid search as the ones with the highest Win Count average (since it remains the underlying objective). The performance of the best agents, and the training parameters used to obtain them are presented respectively in Tables 6.6 and 6.7. We still present their Win Count and corresponding $L2$-Loss.

	Dict-opt		RegNet1		RegNet2	
Training method	CLONALG	GA	CLONALG	GA	CLONALG	GA
Best Avg. Win Count	112.4	36.6	261.8	235.2	250.6	193.1
Absolute Win Rate	32%	10%	75%	67%	72%	55%
Relative Win Rate	36%	12%	83%	75%	90%	69%
Corresponding Avg. L2	13.88	127.24	2.16	2.62	0.96	2.20

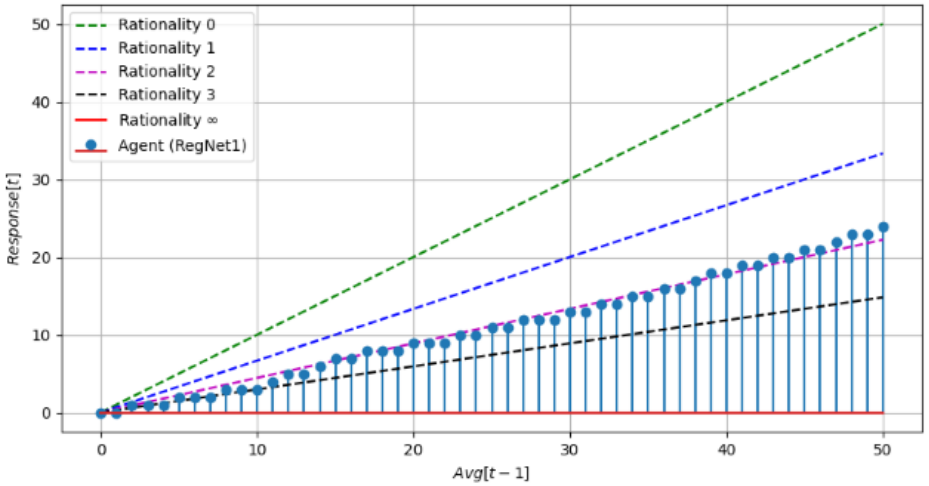
TABLE 6.6: Performance of best agent configurations obtained by grid search in 500 generations ($fitness = L2$)

	Dict-opt		RegNet1		RegNet2	
Training method	CLONALG	GA	CLONALG	GA	CLONALG	GA
pop. size	40	20	40	20	40	40
select. rate	0.5	0.25	0.25	0.5	0.5	0.5
clone rate	0.25	**N.A.**	0.5	**N.A.**	0.25	**N.A.**
$\beta \times mag$	2	**N.A.**	2	**N.A.**	1	**N.A.**
crossover	**N.A.**	M.-P.	**N.A.**	U.	**N.A.**	M.-P.
mut. rate	**N.A.**	0.05	**N.A.**	0.05	**N.A.**	0.05
Elite & (or) rep. rate	0.15	0.15	0.15	0.15	0.15	0.15

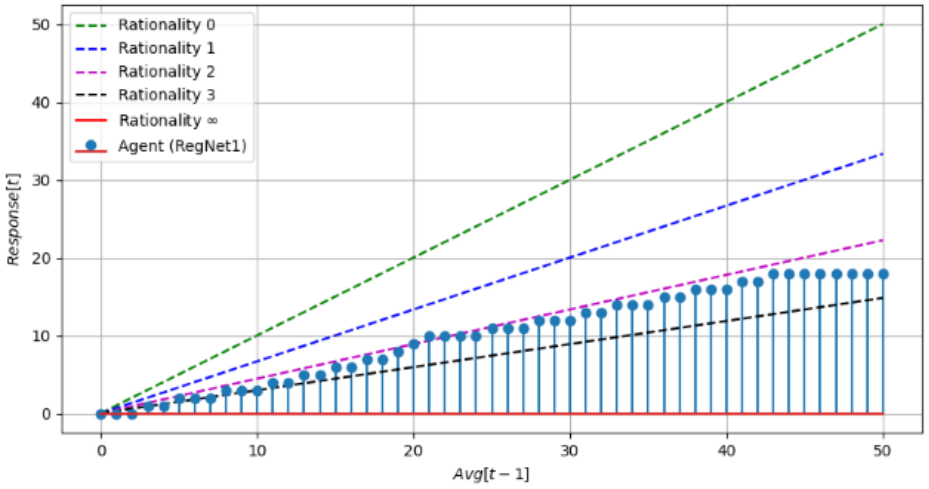
TABLE 6.7: Training hyperparameters used to obtain best configurations in 500 generations ($fitness = L2$)

The CLONALG-trained RegNet1 has the best performance: an absolute win rate of 75% on average - very close to the best performances found previously . Its $L2$-Loss is also quite low, attesting that Loss optimisation implies Win Count optimisation. The GA-trained version has a correct win rate (67%) and $L2$-Loss (2.62), but they are not

quite as good as the latter agent's ones. The responses of both agents - seen in Figure 6.4 - are globally similar, with the same increasing tendency near the level-2 curve, as found previously. However, the GA-trained network seems to underestimate more the target compared to the CLONALG-trained one, likely explaining its slight under-performance.



(A) Response of CLONALG-trained RegNet1

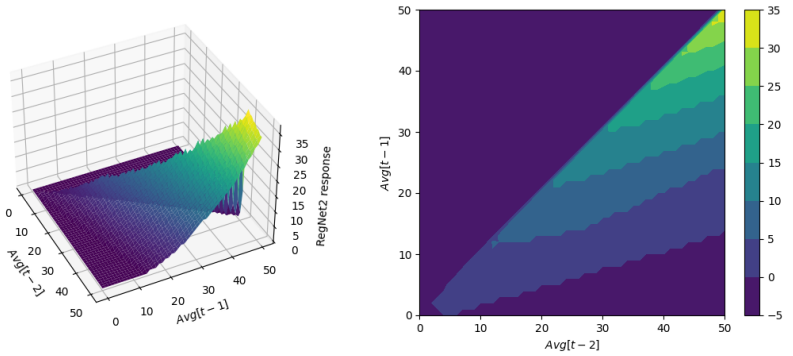


(B) Response of GA-trained RegNet1

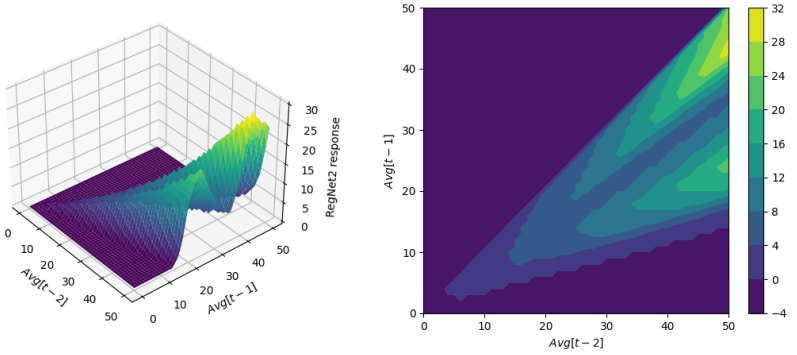
FIGURE 6.4: Response to memory of RegNet1 ($objective = L2$)

Then, we observe that the RegNet2 reaches a relatively excellent performance when trained with CLONALG: 250.6 wins on average, thus a relative win rate of $250.6/280 = 90\%$. Its low Loss 0.96 is also the best one so far, and shows one more time that minimising it enables to be closer globally to the target and win more. On the other hand, the GA-trained network has a subpar performance with 193.1 wins on average.

Comparing their response graphs - in Figure 6.5 - the response surface of the CLONALG-based solution is very regular with increasing steps along the line $Avg[t-1] = Avg[t-2]$, which is intuitively a desirable profile. The GA-based solution possesses a profile that increases in steps as well, however it is split in half by a deep gap. It can be presumed that this gap is the source of bad plays, by underestimating the target in the regions within it. It is possible that this agent didn't converge properly yet, and that additional training generations would contribute to close this gap.



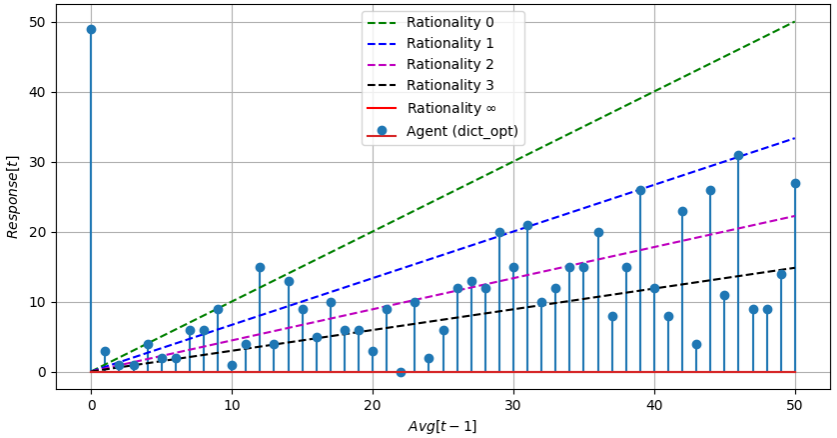
(A) 3-D response and contour plot of CLONALG-trained RegNet2. The section verifying $(Avg[t-1] > Avg[t-2])$ has been removed in the contour plot



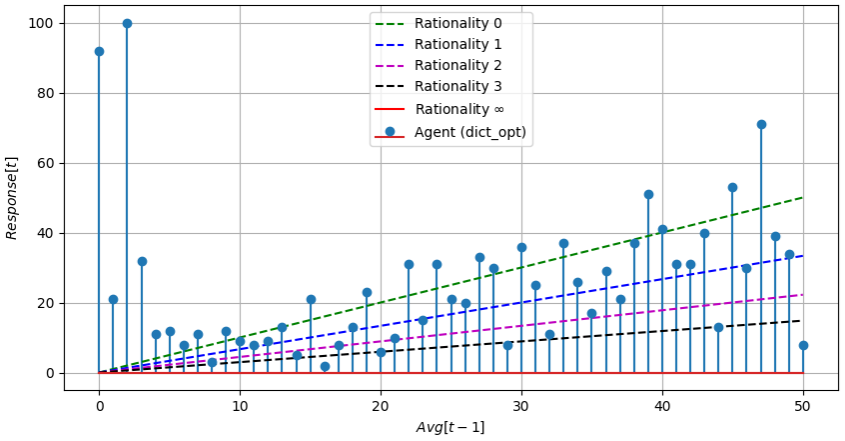
(B) 3-D response and contour plot of GA-trained RegNet2. The section verifying $(Avg[t-1] > Avg[t-2])$ has been removed in the contour plot

FIGURE 6.5: Response surfaces of RegNet2 ($objective = L2$)

Finally, the Dict-opt performances when attempting to minimise the Loss are quite mediocre, in terms of wins or loss values. As shown in their - once again - erratic response profile (Figure 6.6), the agents didn't converge, in 500 generations, to a proper distribution of responses.



(A) Response of CLONALG-trained Dict-opt



(B) Response of GA-trained Dict-opt

FIGURE 6.6: Response to memory of Dict-opt ($objective = L2$)

Like previously, the *pop. size* used to train the best agents is often high, but the other found parameters are more heterogeneous. Some instances still keep same values as previously found ($\beta \times mag = 2$, $mut.\ rate = 0.05$).

6.4 Repeatability of results and similarity test between CLONALG and GA

In this section we do additional tests on the best hyperparameters found for each agent and training paradigm. Using these, training will be run 5 times. The average fitness of the 5 best individuals, standard deviation, maximum and minimum values will be

computed. We still distinguish experiments depending on the fitness optimised (Win Count and $L2$-Loss) by the agents. Results are presented below :

	Dict-opt		RegNet1		RegNet2	
Training method	CLONALG	GA	CLONALG	GA	CLONALG	GA
Avg. WC	224.0	129.8	266.8	257.8	246.0	199.0
σ_{wc}	19.8	9.9	2.5	4.5	4.6	18.1
min. WC	199	112	264	249	241	174
max. WC	245	140	271	262	254	230

TABLE 6.8: Statistics of WC-optimisation convergence on 5 trials

	Dict-opt		RegNet1		RegNet2	
Training method	CLONALG	GA	CLONALG	GA	CLONALG	GA
Avg. L2	81.10	249.91	2.32	3.17	1.43	4.38
σ_{L2}	62.49	92.28	0.47	0.83	0.58	3.78
min. L2	13.90	143.04	1.83	2.26	0.95	1.80
max. L2	176.29	401.55	3.18	4.48	2.52	11.81

TABLE 6.9: Statistics of $L2$-optimisation convergence on 5 trials

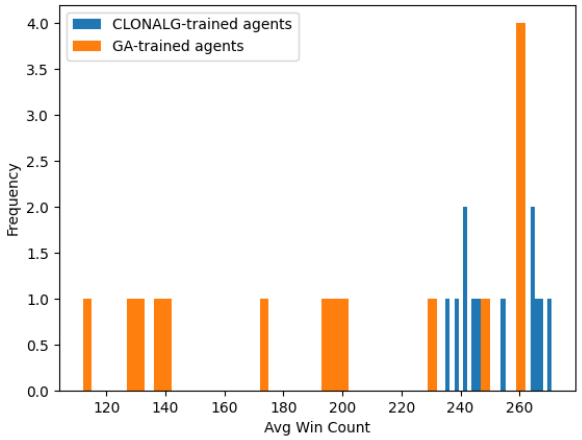
We see that in general Dict-opt is unsafe to train, producing either poor performance (low Win Count or high $L2$-Loss) or too high variance of results. Only the instance where it is trained by CLONALG on the Win Count fitness produces very acceptable results ($Avg.\ Win\ Count = 224$) - close to the one accepted in Table 6.4 - despite a somewhat high variance ($\sigma_{wc} = 19.8$).

For the RegNets, they have in general a very strong performance, trained on either fitness and with any metaheuristic. Only the 2 instances where RegNet2 is trained by GA, are a little subpar. It is a little surprising for the WC-optimising paradigm, as we reported a much higher Win Count in Table 6.4.

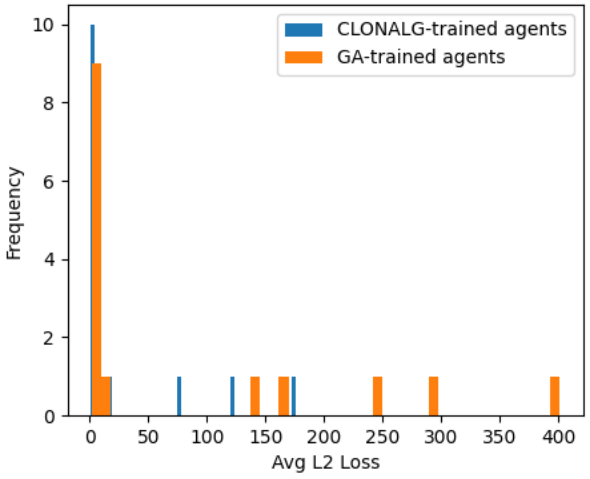
Otherwise, the RegNets' results have low variance and are close - maybe a little worse sometimes though - to the ones reported in Tables 6.4 and 6.6. It means that the

repeatability of results is quite good. On another note, we observe that the RegNets can often converge in 100 or 200 generations, which could enable to reduce training time.

It is then relevant to wonder if CLONALG and GA results are similar (all agents included). For example, we would like to know if they can actually be drawn from the same distribution or not. Converged-performance distribution histograms, where the three types of agents are mixed, are the following :



(A) Converged-best-performance distribution, when optimising WC



(B) Converged-best-performance distribution, when optimising $L2$

FIGURE 6.7: Converged-best-performance distribution of all agents

These distributions cannot be approximated as Gaussian ones, so a T-test may be abusive for measuring similarity between CLONALG and GA [Fay and Proschan, 2010]. However, it is still possible to assess whether GA and CLONALG-converged performances are stochastically equivalent or not ($P(x > y) \approx P(x < y)$). Here, by considering that the CLONALG and GA populations are from independent runs ($3 \times 5 = 15$

runs for each), we will use the Mann-Whitney test to do so [Mann and Whitney, 1947]. The Wilcoxon Test - also called Signed Rank Test - would also have some validity if we consider matched populations [Wilcoxon, 1945]. The null hypothesis of the Mann-Whitney test is **H0**: **GA and CLONALG-trained agents have stochastically a similar performance**. According to the graph shapes, we can formulate the alternative hypothesis of the one-tailed Mann-Whitney test as **H1**: **CLONALG-trained agents have stochastically a better performance than GA-trained ones**. Taking the threshold of significance $\alpha = 0.05$, we compute the p-value of the test on the 2 populations which are the GA-trained and CLONALG-trained performances. Two tests are computed: one test when the performance is the Win Count, and the other when it is the $L2$-Loss.

The one-tailed Mann-Whitney test gives:

- $p_{WC} = 0.041 < \alpha$. Therefore, we can reject **H0** in favour of **H1**: When optimising the Win Count, CLONALG converged agents have stochastically a better performance (greater Win Count) than GA ones.
- $p_{L2} = 0.045 < \alpha$. We can reject **H0** in favour of **H1**: When optimising the $L2$-Loss, CLONALG converged agents have stochastically a better performance (lower Loss) than GA ones.

We however need to emit some reservations about the testing methodology we used. Indeed, we mixed all three types of agents' performance under either the category of CLONALG or GA. Nevertheless, some same-type agents do have very different performances, especially within the Dict-opt class, so they may create imbalance in the distribution of results. For example in the $L2$-optimisation case, removing the Dict-opt agents should certainly highlight more similarity between GA and CLONALG-trained RegNets. Therefore, the test only compares the globality of agents performance, when trained with either CLONALG or GA.

6.5 Hybrid-memory agent: HybridNet

As noted before, RegNet2 has often a relative win rate higher than RegNet1. However, it consistently loses the 2 first idle rounds of each game to create memory, which impacts

significantly its Win Count. Therefore, we propose an hybrid model of agent, called HybridNet, which plays like RegNet1 in the second round (the first one remains idle, during which it plays 50), and then like the RegNet2 from the third one onwards.

This agent would have 2 lists of weights (or a combined list of both), the first one corresponding to the 141 weights of RegNet1, and the second one to the 151 ones of the RegNet2. We choose the weights of the best RegNet1 and RegNet2 found during grid search, i.e. the ones performing respectively 265.6 and 250.6 wins on average. By evaluating 10 times the hybrid agent against the 35 crowds, we observe the following performance :

10-$confrontations\ Avg.\ WC$	282.1
Absolute Win Rate	81%
Relative Win Rate	90%

TABLE 6.10: Performance of HybridNet

The results of HybridNet are excellent, and the best observed so far. With 282.1 wins over 350 possible, it has an absolute win rate of 81% and a relative win rate of 90%, which means that it conserves the relative performance of RegNet2, that was already excellent. This also proves that the RegNet2 outperforms RegNet1 on the 8 last rounds of a game.

6.6 Randomised-levels crowds

The possibility to play the game using only one memory raises the question whether an agent can consistently win or not against crowds playing with random rationality levels (always in $\{0, 1, 2, 3\}$) at each round. This model lets place to dynamic change of rationality, even though it may not always be realistic. In this sub-section, we evaluate the best agents (by average Win Count) found for each paradigm (Dict-opt, RegNet1, RegNet2 and HybridNet) against 35 crowds during 10 rounds, which however change randomly the levels of the 4 players after each round. We repeat 10 confrontations and compute the average Win Count and Win Rate of the agents. The performance of the agents is summarised in Table 6.11 .

	Dict-opt	RegNet1	RegNet2	HybridNet
10-$confrontations$ *Avg. WC*	245.4	247.6	209.7	242.7
Absolute Win Rate	70%	71%	60%	69%

TABLE 6.11: Performance of each model against randomised-levels crowds

As we could have expected, Dict-opt and RegNet1 perform the best, with respectively 245.4 and 247.6 wins on average, over the 350 possible, since they are memory-1 agents which don't correlate 2 memories to fire a response. These are very correct win rates, which are however slightly below the results obtained in static-rationality crowds (respectively 72% and 76%). In fact this is not that surprising, because they were trained against simulated crowds which followed a rational pattern. It means that when certain crowds reached $Avg[t-1]$, there was often a limited set of correct responses at t, that would do well against any of them. When playing against randomised-level players, given $Avg[t-1]$, the set of possible responses at t is wider and depends on which rational levels the crowds will be taking at t.

RegNet2 has an average performance, which was expected. Indeed, using 2 round averages allows the agent to reproduce the tendency of play of the crowd, or rather to counter it. However, against randomised-levels crowds, the crowd's tendency of play can be very variable at each pair of successive time steps $t-1$ and t. Therefore, $Avg[t-2]$ and $Avg[t-1]$ may lead the agent response in a region (cf Figures 6.2 and 6.5) which doesn't correspond to the real state of the game.

What is surprising however, is the very good performance of HybridNet, slightly below but still similar to the ones of Dict-opt and RegNet1. Indeed, this agent corresponds to RegNet1 during the round 2, and to RegNet2 during the rounds $3-10$, to which we could expect it to have the same performance as the latter. In fact, this outperformance is likely due to the second round, where by adopting the behaviour of RegNet1, the agent shifts the average towards low values (instead of playing 50, which rather lifts the average), in a region where it has a consistent performance regardless of the crowd being played. In such a region, $Avg[t-1]$ has more influence than $Avg[t-2]$ in the decision - as seen in Figure 6.5a - allowing to respond similarly to a memory-1 agent like Dict-opt and RegNet1. This contrast between RegNet2 and HybridNet can be seen in the transcripts below (Figure 6.8) of some games played between the agents and

randomised-levels crowds, notably from Round 1. The agent's play is systematically the last one in the choices list (5^{th} element or index 4).

```
--------------------------------------
RANDOM CROWD
Round 0
Choices = [9, 33, 34, 51, 50]
Average = 35.4
Target = 23.6
Winner(s) = [1]
--------------------------------------
Round 1
Choices = [16, 15, 16, 10, 50]
Average = 21.4
Target = 14.27
Winner(s) = [1]
--------------------------------------
Round 2
Choices = [9, 21, 14, 21, 8]
Average = 14.6
Target = 9.73
Winner(s) = [0]
--------------------------------------
Round 3
Choices = [7, 4, 4, 10, 7]
Average = 6.4
Target = 4.27
Winner(s) = [1 2]
--------------------------------------
```

(A) Transcript of some rounds played by RegNet2 (index = 4) against randomised-levels crowds

```
--------------------------------------
RANDOM CROWD
Round 0
Choices = [29, 13, 30, 1, 50]
Average = 24.6
Target = 16.4
Winner(s) = [1]
--------------------------------------
Round 1
Choices = [16, 11, 11, 17, 11]
Average = 13.2
Target = 8.8
Winner(s) = [1 2 4]
--------------------------------------
Round 2
Choices = [13, 9, 13, 6, 6]
Average = 9.4
Target = 6.27
Winner(s) = [3 4]
--------------------------------------
Round 3
Choices = [4, 4, 9, 4, 4]
Average = 5.0
Target = 3.33
Winner(s) = [0 1 3 4]
--------------------------------------
```

(B) Transcript of some rounds played by HybridNet (index = 4) against randomised-levels crowds

FIGURE 6.8: Transcripts of plays of RegNet2 and HybridNet (index = 4) against randomised-levels crowds

6.7 Extrapolated plays against larger crowds

Our framework fixed the number of players at $N = 5$, in which we have succeeded in extracting agents with very good performance. In this sub-section, we investigate the following question: can the agents trained in size-5 crowds conserve their performance against groups of larger size, but which still adopt rationality levels in $[\![0, 3]\!]$? For the sake of brevity, we only evaluate the agent HybridNet, 10 times against crowds of size 6, 10 and 20. We represent its results in the Table 6.12 below, in which we still display the performance against the size-5 crowd.

Crowd size	5	6	10	20
Nb. of possible crowds	35	56	220	1540
Absolute Win Rate	81%	79%	77%	74%

TABLE 6.12: Win rates of HybridNet against crowds of different sizes

We can see that the agent's performance still remains correct, always above 70% win rate, but tends to decrease the more there are players in the crowd. This means that training against a particular crowd size, it is still possible to extrapolate accurate plays against other crowds, but this approximation loses in accuracy the larger they become. This observation can be demonstrated by majoring and minoring the agent prediction or target. To simplify, we consider predictions to be real-valued. Let N be the number of players, and $A[t]$ and $P_i[t]$ the respective plays of the agent and player i (of rationality k_i) at t. Assuming the agent plays perfectly, we have $A[t] = Target[t]$. Thus :

$$\begin{aligned} Target[t] &= p.\frac{\sum_{i=1}^{N-1} P_i[t] + A[t]}{N} \\ \iff A[t] &= p.\frac{\sum_{i=1}^{N-1} P_i[t] + A[t]}{N} \\ \iff A[t] &= \frac{N}{N-p}.p.\frac{\sum_{i=1}^{N-1} P_i[t]}{N} \\ &= \frac{p}{N-p}.Avg[t-1]\sum_{i=1}^{N-1} p^{k_i} \end{aligned} \tag{6.4}$$

By considering that $p^3 \leq p^{k_i} \leq 1$, and substituting p to $2/3$:

$$\begin{aligned} & \frac{p.(N-1)}{N-p}.p^3 \leq \frac{A[t]}{Avg[t-1]} \leq \frac{p.(N-1)}{N-p}. \\ \iff \quad & \frac{2N-2}{3N-2}.\frac{8}{27} \leq \frac{A[t]}{Avg[t-1]} \leq \frac{2N-2}{3N-2} \\ \iff \quad & (1-\frac{N}{3N-2}).\frac{8}{27} \leq \frac{A[t]}{Avg[t-1]} \leq 1-\frac{N}{3N-2} \end{aligned} \tag{6.5}$$

We can see that each majorant and minorant is an upperly-bounded increasing sequence. By passing the inequality to the limit, we have :

$$\begin{gathered} When\ N \to +\infty, \\ 16/81 \approx 0.198 \leq \frac{A[t]}{Avg[t-1]} \leq 2/3 \approx 0.667 \end{gathered} \tag{6.6}$$

When $N = 5$, we have approximately:

$$0.182 \leq \frac{A[t]}{Avg[t-1]} \leq 0.615 \tag{6.7}$$

When facing a crowd with an arbitrarily high size, plays in $\Big[0.182, 0.198\Big].Avg[t-1]$ are futile, while the ones in $\Big[0.615, 0.667\Big].Avg[t-1]$ are not reachable. Having trained against crowds of size N, we therefore cannot expect to extrapolate perfect plays and - similar performances - against larger crowds, even though it is possible to give a good approximation against sizes close to N. The size-6-crowd interval of perfect plays is $\Big[0.185, 0.625\Big].Avg[t-1]$, which is close to $\Big[0.182, 0.615\Big].Avg[t-1]$ for example, so we can expect that HybridNet covers most of the former interval. From the graph in Figure 6.9 displaying the evolution of the bounds of perfect plays, it can be conjectured that training again size-20 crowds is a good trade-off for extrapolating plays against - a priori - any larger crowd. This conjecture would need experimentation to be proved empirically.

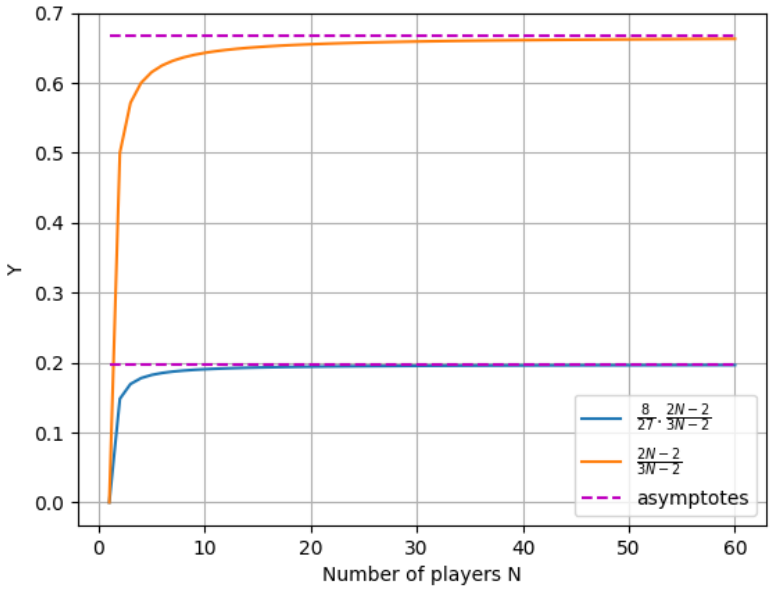


FIGURE 6.9: Evolution of lower and upper bounds of perfect predictions in function of the number of players N in the crowd

Chapter 7

Conclusion and Future Works

7.1 Conclusion

We exhibit the possibility to train - using classic implementations of CLONALG and GA - agents performing very well against level-k players in the KBC, in an "all-rounder" model, i.e. the agent plays against each combination of rationality levels. The two paradigms studied, dictionary optimisation and Neuroevolution, use (short-term) memory to launch response. In general, optimising the Win Count has proven to give better results than optimising the $L2$-Loss, except for the memory-2 agent. The hypothesis that a more steep integer-valued fitness prevents convergence does not seem relevant in this case. Moreover, a Mann-Whitney test - taking $\alpha = 0.05$ - on repeated experiments tends to demonstrate that CLONALG produces generally better performance than GA - all agents included - which can be due to the specificity of the problem.

Dict-opt has produced one well-performing agent (72% win rate) being trained by CLONALG, but seems to be unsafe to train as we observe quite high variance of repeated results, or poor convergence in general. The brutal nature of the mutation might be at fault, and solutions would be to either increase the number of training generations or using softer mutations like Gaussian shrinkage.

The 2 neuroevolved networks have produced excellent agents, reaching respective peak win rates of 76% and 72% using CLONALG. It always outperformed GA, which is not a result guaranteed by literature [He et al., 2005, Ulker and Ulker, 2012], and may

be due to the set of parameters tested, or just the nature of the problem itself. GA-trained RegNets have nonetheless not so far performance from CLONALG-trained ones in general. Repeatability of results is also good when they are trained multiple times with CLONALG and GA, and both methods are consistent as well, as confirmed by the low variance of converged performances.

Their converged response profiles are satisfactory as well, being smooth and increasing gradually, in contrast to the more erratic profile of Dict-opt. We also see that in the memory-1 case, the optimal profile is often close to a Level-2, sometimes 3, gameplay. It tends to match the observation of Nagel [1995] that the Level-2 was often an optimal playtype in her social experiments. The neuroevolved agents (and Dict-opt as well) prove to be able to shift towards the theoretical Nash Equilibrium 0, which demonstrates long-term consistency. An hybrid agent, HybridNet, behaving like RegNet1 in the second round, and RegNet2 afterwards, has an improved and exceptional performance of 81%, showing that using 2 memories is indeed more beneficial than using only one. The main drawback was that 2/10 rounds were systematically sacrificed by RegNet2 to constitute the first memories.

Additional tests show that memory-1 agents (Dict-opt and RegNet1) remain consistent - a little less though - against players that change randomly levels between rounds, which is not that surprising given that they use a single memory to launch response. The performance of HybridNet is similar and tends to show that by lowering immediately the average of plays, its response is located in a region where $Avg[t-1]$ has more influence than $Avg[t-2]$. Extrapolating plays against larger crowds is also possible, but loses in precision the bigger crowds become. A rough calculation of the perfect play's bounds leads to conjecture that training against 20 players should be enough to generalise plays against any larger crowds. This remains to be tested though, and may be computationally intensive since there would be 1540 different crowds to confront.

To conclude, this study highlights that, by discovering strategies with great consistency and tending to be close to theoretical and empirical results, CLONALG is an efficient metaheuristic, able to provide better results than GA in this case. Results are encouraging to investigate AIS more in the future, alongside GA, PSO and other optimisation techniques, notably in the field of decision taking and Neuroevolution.

7.2 Future Works

Even though excellent agents have been extracted using the Level-k model, there is no guarantee a priori that such agents would do well against humans. In the first place, this study didn't really seek to make an agent able to play against humans, but rather to assess if metaheuristics - and mainly CLONALG - could discover strategies against the acknowledged level-k pattern of play. However, such an agent could be used as an evaluator to verify whether or not humans really adopt the level-k model. If it keeps a win rate above 70% against humans, we could for example consider with great confidence that the Level-k model is adopted. Therefore, a future experiment could be to gather many groups of 4 people to play with HybridNet, and compute its average performance. The game and the agent would be hosted on a website ideally, and it could be more appropriate to deceive the participants into thinking that they play against only humans, to not bias their decision.

In terms of practical consequences, optimising the KBC doesn't mean much on its own. Nevertheless, the methodology used could be re-applied in other contexts where an agent has to take a decision in an environment with which it interacts. Keynes [1936] made an analogy comparing stock market dynamics to the ones of the Beauty Contest, since investors try to anticipate others to sell their stocks at the most advantageous time. It would be naive to think that our method could directly extrapolate stock market prediction, however it is worth investigating if this framework or other - similar or not - real-world competitions could be modelled and solved using the methods developed, and notably Neuroevolution.

And last but not least, an alternative immune-inspired NEAT, which would rely on hypermutation and not crossover, could be investigated to propose another search method. In their original paper, Stanley and Miikkulainen [2002] did find that a non-crossing NEAT had a failure rate of 0% in the Double Balancing Pole problem, just the same as the crossing NEAT. The difference was that the crossing NEAT found a solution with less evaluations. This is still encouraging to adapt a CLONALG-version of NEAT, all the more that it is a very popular Neuroevolved technique.

Appendix A

Partial examples of games

We display partial examples of plays (from the 3^{rd} to 6^{th} round) between best agents and the crowd of levels $(0, 1, 2, 3)$. The agent's play is always the last in the "list of choices" (index 4).

```
--------------------------------------
Round 2
Choices = [19, 13, 8, 6, 8]
Average = 10.8
Target = 7.2
Winner(s) = [2 4]
--------------------------------------
Round 3
Choices = [11, 8, 5, 4, 4]
Average = 6.4
Target = 4.27
Winner(s) = [3 4]
--------------------------------------
Round 4
Choices = [6, 4, 3, 2, 2]
Average = 3.4
Target = 2.27
Winner(s) = [3 4]
--------------------------------------
Round 5
Choices = [4, 2, 1, 1, 1]
Average = 1.8
Target = 1.2
Winner(s) = [2 3 4]
--------------------------------------
```

(A) Plays with RegNet1, trained with CLONALG on $L2$ fitness

```
--------------------------------------
Round 2
Choices = [20, 13, 9, 6, 9]
Average = 11.4
Target = 7.6
Winner(s) = [2 4]
--------------------------------------
Round 3
Choices = [11, 7, 5, 3, 5]
Average = 6.2
Target = 4.13
Winner(s) = [2 4]
--------------------------------------
Round 4
Choices = [6, 5, 3, 2, 2]
Average = 3.6
Target = 2.4
Winner(s) = [3 4]
--------------------------------------
Round 5
Choices = [4, 3, 2, 1, 2]
Average = 2.4
Target = 1.6
Winner(s) = [2 4]
--------------------------------------
```

(B) Plays with RegNet1, trained with CLONALG on WC fitness

```
--------------------------------------
Round 2
Choices = [20, 13, 8, 5, 8]
Average = 10.8
Target = 7.2
Winner(s) = [2 4]
--------------------------------------
Round 3
Choices = [11, 7, 5, 3, 4]
Average = 6.0
Target = 4.0
Winner(s) = [4]
--------------------------------------
Round 4
Choices = [6, 4, 3, 1, 2]
Average = 3.2
Target = 2.13
Winner(s) = [4]
--------------------------------------
Round 5
Choices = [3, 2, 2, 1, 1]
Average = 1.8
Target = 1.2
Winner(s) = [3 4]
--------------------------------------
```

(C) Plays with RegNet1, trained with GA on $L2$ fitness

```
--------------------------------------
Round 2
Choices = [17, 11, 8, 5, 8]
Average = 9.8
Target = 6.53
Winner(s) = [2 4]
--------------------------------------
Round 3
Choices = [10, 6, 5, 3, 3]
Average = 5.4
Target = 3.6
Winner(s) = [3 4]
--------------------------------------
Round 4
Choices = [5, 4, 2, 2, 2]
Average = 3.0
Target = 2.0
Winner(s) = [2 3 4]
--------------------------------------
Round 5
Choices = [3, 2, 2, 1, 1]
Average = 1.8
Target = 1.2
Winner(s) = [3 4]
--------------------------------------
```

(D) Plays with RegNet1, trained with GA on WC fitness

FIGURE A.1: Transcripts of plays of RegNet1 against the crowd $(0, 1, 2, 3)$. It plays last in the list "Choices"

```
---------------------------------------
Round 2
Choices = [29, 20, 13, 9, 12]
Average = 16.6
Target = 11.07
Winner(s) = [4]
---------------------------------------
Round 3
Choices = [17, 11, 8, 5, 6]
Average = 9.4
Target = 6.27
Winner(s) = [4]
---------------------------------------
Round 4
Choices = [9, 6, 4, 3, 3]
Average = 5.0
Target = 3.33
Winner(s) = [3 4]
---------------------------------------
Round 5
Choices = [5, 4, 2, 1, 2]
Average = 2.8
Target = 1.87
Winner(s) = [2 4]
---------------------------------------
```

(A) Plays with RegNet2, trained with CLONALG on $L2$ fitness

```
---------------------------------------
Round 2
Choices = [30, 20, 13, 9, 8]
Average = 16.0
Target = 10.67
Winner(s) = [3]
---------------------------------------
Round 3
Choices = [16, 11, 7, 4, 4]
Average = 8.4
Target = 5.6
Winner(s) = [2]
---------------------------------------
Round 4
Choices = [8, 5, 4, 2, 3]
Average = 4.4
Target = 2.93
Winner(s) = [4]
---------------------------------------
Round 5
Choices = [4, 3, 2, 1, 2]
Average = 2.4
Target = 1.6
Winner(s) = [2 4]
---------------------------------------
```

(B) Plays with RegNet2, trained with CLONALG on WC fitness

```
---------------------------------------
Round 2
Choices = [26, 17, 12, 7, 12]
Average = 14.8
Target = 9.87
Winner(s) = [2 4]
---------------------------------------
Round 3
Choices = [15, 10, 7, 5, 8]
Average = 9.0
Target = 6.0
Winner(s) = [2 3]
---------------------------------------
Round 4
Choices = [9, 6, 4, 3, 5]
Average = 5.4
Target = 3.6
Winner(s) = [2]
---------------------------------------
Round 5
Choices = [5, 3, 3, 1, 2]
Average = 2.8
Target = 1.87
Winner(s) = [4]
---------------------------------------
```

(C) Plays with RegNet2, trained with GA on $L2$ fitness

```
---------------------------------------
Round 2
Choices = [25, 17, 11, 7, 12]
Average = 14.4
Target = 9.6
Winner(s) = [2]
---------------------------------------
Round 3
Choices = [14, 9, 6, 4, 5]
Average = 7.6
Target = 5.07
Winner(s) = [4]
---------------------------------------
Round 4
Choices = [8, 5, 3, 2, 3]
Average = 4.2
Target = 2.8
Winner(s) = [2 4]
---------------------------------------
Round 5
Choices = [4, 3, 2, 1, 2]
Average = 2.4
Target = 1.6
Winner(s) = [2 4]
---------------------------------------
```

(D) Plays with RegNet2, trained with GA on WC fitness

FIGURE A.2: Transcripts of plays of RegNet2 against the crowd $(0, 1, 2, 3)$. It plays last in the list "Choices"

```
---------------------------------------
Round 2
Choices = [18, 12, 7, 5, 6.0]
Average = 9.6
Target = 6.4
Winner(s) = [4]
---------------------------------------
Round 3
Choices = [10, 6, 5, 3, 1.0]
Average = 5.0
Target = 3.33
Winner(s) = [3]
---------------------------------------
Round 4
Choices = [5, 3, 3, 1, 2.0]
Average = 2.8
Target = 1.87
Winner(s) = [4]
---------------------------------------
Round 5
Choices = [3, 2, 2, 1, 1.0]
Average = 1.8
Target = 1.2
Winner(s) = [3 4]
---------------------------------------
```

(A) Plays with Dict-opt, trained with CLONALG on $L2$ fitness

```
---------------------------------------
Round 2
Choices = [15, 11, 7, 5, 8.0]
Average = 9.2
Target = 6.13
Winner(s) = [2]
---------------------------------------
Round 3
Choices = [9, 6, 4, 2, 3.0]
Average = 4.8
Target = 3.2
Winner(s) = [4]
---------------------------------------
Round 4
Choices = [6, 3, 2, 2, 2.0]
Average = 3.0
Target = 2.0
Winner(s) = [2 3 4]
---------------------------------------
Round 5
Choices = [3, 2, 1, 1, 1.0]
Average = 1.6
Target = 1.07
Winner(s) = [2 3 4]
---------------------------------------
```

(B) Plays with Dict-opt, trained with CLONALG on WC fitness

```
---------------------------------------
Round 2
Choices = [20, 13, 9, 5, 6.0]
Average = 10.6
Target = 7.07
Winner(s) = [4]
---------------------------------------
Round 3
Choices = [11, 7, 5, 3, 8.0]
Average = 6.8
Target = 4.53
Winner(s) = [2]
---------------------------------------
Round 4
Choices = [7, 5, 3, 2, 11.0]
Average = 5.6
Target = 3.73
Winner(s) = [2]
---------------------------------------
Round 5
Choices = [6, 4, 3, 2, 8.0]
Average = 4.6
Target = 3.07
Winner(s) = [2]
---------------------------------------
```

(C) Plays with Dict-opt, trained with GA on $L2$ fitness

```
---------------------------------------
Round 2
Choices = [30, 20, 13, 9, 9.0]
Average = 16.2
Target = 10.8
Winner(s) = [3 4]
---------------------------------------
Round 3
Choices = [16, 11, 7, 4, 69.0]
Average = 21.4
Target = 14.27
Winner(s) = [0]
---------------------------------------
Round 4
Choices = [21, 14, 9, 6, 4.0]
Average = 10.8
Target = 7.2
Winner(s) = [3]
---------------------------------------
Round 5
Choices = [11, 7, 5, 3, 3.0]
Average = 5.8
Target = 3.87
Winner(s) = [3 4]
---------------------------------------
```

(D) Plays with Dict-opt, trained with GA on WC fitness

FIGURE A.3: Transcripts of plays of Dict-opt against the crowd $(0, 1, 2, 3)$. It plays last in the list "Choices"

Appendix B

Hyperparameter investigation tables

The hyperparameter encoding for CLONALG optimisation is the following :
$(pop.\ size, select.\ size, clone\ rate, \beta \times mag, replace\ rate)$.

For GA, it is :
$(pop.\ size, select.\ size, cross.\ bool, mut.\ rate, elite\ \&\ replace\ rate)$.

$cross.\ bool = 0$ indicates Uniform crossover, whereas $cross.\ bool = 1$ indicates Middle-point crossover.

We may note that the normalised $L2$ values presented are rounded to 1 decimal, which can explain why values reported in Tables 6.4 and 6.6 can be a little different (rounded to 2 decimals).

	Hyperparameters	Avg Win Count	Avg L2 Loss
0	(10, 0.25, 0.5, 0.5, 0.05)	49.2	1508.3
1	(10, 0.25, 0.5, 0.5, 0.15)	49.5	1587.9
2	(10, 0.25, 0.5, 1, 0.05)	57.7	1634.4
3	(10, 0.25, 0.5, 1, 0.15)	49.7	1743.9
4	(10, 0.25, 0.5, 2, 0.05)	87.1	1907.0
5	(10, 0.25, 0.5, 2, 0.15)	113.1	748.8
6	(10, 0.5, 0.25, 0.5, 0.05)	63.6	1798.4
7	(10, 0.5, 0.25, 0.5, 0.15)	56.2	1761.9
8	(10, 0.5, 0.25, 1, 0.05)	60.8	1927.8
9	(10, 0.5, 0.25, 1, 0.15)	70.5	1492.4
10	(10, 0.5, 0.25, 2, 0.05)	104.1	1219.0
11	(10, 0.5, 0.25, 2, 0.15)	137.9	2270.8
12	(20, 0.25, 0.5, 0.5, 0.05)	78.9	1179.1
13	(20, 0.25, 0.5, 0.5, 0.15)	62.5	1424.2
14	(20, 0.25, 0.5, 1, 0.05)	67.8	1738.1
15	(20, 0.25, 0.5, 1, 0.15)	72.8	1265.5
16	(20, 0.25, 0.5, 2, 0.05)	112.8	1576.7
17	(20, 0.25, 0.5, 2, 0.15)	134.6	1662.5
18	(20, 0.5, 0.25, 0.5, 0.05)	77.0	1182.4
19	(20, 0.5, 0.25, 0.5, 0.15)	57.3	1737.7
20	(20, 0.5, 0.25, 1, 0.05)	85.5	1970.0
21	(20, 0.5, 0.25, 1, 0.15)	89.2	1868.2
22	(20, 0.5, 0.25, 2, 0.05)	157.8	897.7
23	(20, 0.5, 0.25, 2, 0.15)	155.9	1216.9
24	(40, 0.25, 0.5, 0.5, 0.05)	73.1	1737.9
25	(40, 0.25, 0.5, 0.5, 0.15)	68.1	1465.8
26	(40, 0.25, 0.5, 1, 0.05)	99.0	1181.5
27	(40, 0.25, 0.5, 1, 0.15)	106.1	1324.9
28	(40, 0.25, 0.5, 2, 0.05)	223.9	74.3
29	(40, 0.25, 0.5, 2, 0.15)	191.4	499.4
30	(40, 0.5, 0.25, 0.5, 0.05)	73.7	1507.2
31	(40, 0.5, 0.25, 0.5, 0.15)	72.5	1843.4
32	(40, 0.5, 0.25, 1, 0.05)	123.6	1096.7
33	(40, 0.5, 0.25, 1, 0.15)	106.5	1171.0
34	(40, 0.5, 0.25, 2, 0.05)	252.6	19.4
35	(40, 0.5, 0.25, 2, 0.15)	176.4	1694.6

TABLE B.1: Dict-opt (paradigm: CLONALG, fitness: WC) hyperparameter investigation

	Hyperparameters	Avg Win Count	Avg L2 Loss
0	(10, 0.25, 0, 0.05, 0.15)	82.0	1336.9
1	(10, 0.25, 0, 0.15, 0.05)	64.0	1666.3
2	(10, 0.25, 1, 0.05, 0.15)	65.8	1901.9
3	(10, 0.25, 1, 0.15, 0.05)	44.2	1795.9
4	(10, 0.5, 0, 0.05, 0.15)	102.5	1110.8
5	(10, 0.5, 0, 0.15, 0.05)	49.0	2030.4
6	(10, 0.5, 1, 0.05, 0.15)	56.7	1542.0
7	(10, 0.5, 1, 0.15, 0.05)	59.7	1295.5
8	(20, 0.25, 0, 0.05, 0.15)	89.1	1004.6
9	(20, 0.25, 0, 0.15, 0.05)	59.0	1460.6
10	(20, 0.25, 1, 0.05, 0.15)	105.7	1575.8
11	(20, 0.25, 1, 0.15, 0.05)	59.4	1947.6
12	(20, 0.5, 0, 0.05, 0.15)	101.4	916.4
13	(20, 0.5, 0, 0.15, 0.05)	71.6	1901.6
14	(20, 0.5, 1, 0.05, 0.15)	94.6	1093.5
15	(20, 0.5, 1, 0.15, 0.05)	77.8	1883.9
16	(40, 0.25, 0, 0.05, 0.15)	122.1	810.6
17	(40, 0.25, 0, 0.15, 0.05)	73.4	1917.6
18	(40, 0.25, 1, 0.05, 0.15)	107.6	1304.5
19	(40, 0.25, 1, 0.15, 0.05)	72.3	1806.1
20	(40, 0.5, 0, 0.05, 0.15)	123.0	1246.6
21	(40, 0.5, 0, 0.15, 0.05)	77.9	1612.6
22	(40, 0.5, 1, 0.05, 0.15)	101.2	1496.1
23	(40, 0.5, 1, 0.15, 0.05)	60.0	1892.8

Table B.2: Dict-opt (paradigm: GA, fitness: WC) hyperparameter investigation

	Hyperparameters	Avg Win Count	Avg L2 Loss
0	(10, 0.25, 0.5, 0.5, 0.05)	134.9	31.5
1	(10, 0.25, 0.5, 0.5, 0.15)	134.6	30.4
2	(10, 0.25, 0.5, 1, 0.05)	223.4	7.3
3	(10, 0.25, 0.5, 1, 0.15)	228.4	5.4
4	(10, 0.25, 0.5, 2, 0.05)	249.3	3.0
5	(10, 0.25, 0.5, 2, 0.15)	237.2	14.0
6	(10, 0.5, 0.25, 0.5, 0.05)	181.1	31.2
7	(10, 0.5, 0.25, 0.5, 0.15)	155.8	27.1
8	(10, 0.5, 0.25, 1, 0.05)	261.7	3.9
9	(10, 0.5, 0.25, 1, 0.15)	230.6	5.9
10	(10, 0.5, 0.25, 2, 0.05)	249.2	6.2
11	(10, 0.5, 0.25, 2, 0.15)	257.5	2.2
12	(20, 0.25, 0.5, 0.5, 0.05)	179.1	31.4
13	(20, 0.25, 0.5, 0.5, 0.15)	134.8	31.2
14	(20, 0.25, 0.5, 1, 0.05)	260.6	2.9
15	(20, 0.25, 0.5, 1, 0.15)	258.4	4.7
16	(20, 0.25, 0.5, 2, 0.05)	259.2	2.3
17	(20, 0.25, 0.5, 2, 0.15)	265.6	2.1
18	(20, 0.5, 0.25, 0.5, 0.05)	183.0	30.3
19	(20, 0.5, 0.25, 0.5, 0.15)	230.4	4.0
20	(20, 0.5, 0.25, 1, 0.05)	261.6	2.2
21	(20, 0.5, 0.25, 1, 0.15)	256.9	4.4
22	(20, 0.5, 0.25, 2, 0.05)	250.2	2.9
23	(20, 0.5, 0.25, 2, 0.15)	256.2	2.9
24	(40, 0.25, 0.5, 0.5, 0.05)	230.2	11.7
25	(40, 0.25, 0.5, 0.5, 0.15)	243.6	6.5
26	(40, 0.25, 0.5, 1, 0.05)	256.4	2.1
27	(40, 0.25, 0.5, 1, 0.15)	261.3	4.7
28	(40, 0.25, 0.5, 2, 0.05)	254.3	2.5
29	(40, 0.25, 0.5, 2, 0.15)	260.7	4.9
30	(40, 0.5, 0.25, 0.5, 0.05)	243.2	8.4
31	(40, 0.5, 0.25, 0.5, 0.15)	219.4	4.7
32	(40, 0.5, 0.25, 1, 0.05)	258.1	2.8
33	(40, 0.5, 0.25, 1, 0.15)	263.2	1.9
34	(40, 0.5, 0.25, 2, 0.05)	258.5	2.7
35	(40, 0.5, 0.25, 2, 0.15)	260.5	2.7

TABLE B.3: RegNet1 (paradigm: CLONALG, fitness: WC) hyperparameter investigation

	Hyperparameters	Avg Win Count	Avg L2 Loss
0	(10, 0.25, 0, 0.05, 0.15)	246.8	5.4
1	(10, 0.25, 0, 0.15, 0.05)	253.6	2.7
2	(10, 0.25, 1, 0.05, 0.15)	243.1	15.2
3	(10, 0.25, 1, 0.15, 0.05)	209.3	23.7
4	(10, 0.5, 0, 0.05, 0.15)	257.3	2.4
5	(10, 0.5, 0, 0.15, 0.05)	211.3	29.4
6	(10, 0.5, 1, 0.05, 0.15)	256.7	4.0
7	(10, 0.5, 1, 0.15, 0.05)	210.7	3.4
8	(20, 0.25, 0, 0.05, 0.15)	258.1	2.4
9	(20, 0.25, 0, 0.15, 0.05)	225.3	3.5
10	(20, 0.25, 1, 0.05, 0.15)	260.3	2.4
11	(20, 0.25, 1, 0.15, 0.05)	245.8	7.6
12	(20, 0.5, 0, 0.05, 0.15)	253.0	3.7
13	(20, 0.5, 0, 0.15, 0.05)	240.6	2.0
14	(20, 0.5, 1, 0.05, 0.15)	254.7	2.0
15	(20, 0.5, 1, 0.15, 0.05)	243.4	7.2
16	(40, 0.25, 0, 0.05, 0.15)	263.3	2.2
17	(40, 0.25, 0, 0.15, 0.05)	258.1	3.1
18	(40, 0.25, 1, 0.05, 0.15)	262.2	2.0
19	(40, 0.25, 1, 0.15, 0.05)	252.3	3.6
20	(40, 0.5, 0, 0.05, 0.15)	256.7	2.7
21	(40, 0.5, 0, 0.15, 0.05)	228.8	4.2
22	(40, 0.5, 1, 0.05, 0.15)	251.4	6.2
23	(40, 0.5, 1, 0.15, 0.05)	252.8	2.2

TABLE B.4: RegNet1 (paradigm: GA, fitness: WC) hyperparameter investigation

	Hyperparameters	Avg Win Count	Avg L2 Loss
0	(10, 0.25, 0.5, 0.5, 0.05)	114.8	27.2
1	(10, 0.25, 0.5, 0.5, 0.15)	114.4	27.7
2	(10, 0.25, 0.5, 1, 0.05)	115.2	27.1
3	(10, 0.25, 0.5, 1, 0.15)	113.9	26.9
4	(10, 0.25, 0.5, 2, 0.05)	171.4	6.6
5	(10, 0.25, 0.5, 2, 0.15)	151.5	132.6
6	(10, 0.5, 0.25, 0.5, 0.05)	120.2	77.4
7	(10, 0.5, 0.25, 0.5, 0.15)	113.8	27.0
8	(10, 0.5, 0.25, 1, 0.05)	147.0	19.7
9	(10, 0.5, 0.25, 1, 0.15)	114.5	27.0
10	(10, 0.5, 0.25, 2, 0.05)	202.5	10.7
11	(10, 0.5, 0.25, 2, 0.15)	195.1	13.9
12	(20, 0.25, 0.5, 0.5, 0.05)	113.9	27.7
13	(20, 0.25, 0.5, 0.5, 0.15)	120.9	26.8
14	(20, 0.25, 0.5, 1, 0.05)	121.0	27.6
15	(20, 0.25, 0.5, 1, 0.15)	146.7	16.3
16	(20, 0.25, 0.5, 2, 0.05)	149.7	27.7
17	(20, 0.25, 0.5, 2, 0.15)	212.7	17.9
18	(20, 0.5, 0.25, 0.5, 0.05)	112.7	26.7
19	(20, 0.5, 0.25, 0.5, 0.15)	147.6	38.5
20	(20, 0.5, 0.25, 1, 0.05)	206.8	3.0
21	(20, 0.5, 0.25, 1, 0.15)	242.3	2.1
22	(20, 0.5, 0.25, 2, 0.05)	208.8	22.5
23	(20, 0.5, 0.25, 2, 0.15)	239.7	2.5
24	(40, 0.25, 0.5, 0.5, 0.05)	113.1	27.3
25	(40, 0.25, 0.5, 0.5, 0.15)	121.3	27.1
26	(40, 0.25, 0.5, 1, 0.05)	167.0	51.0
27	(40, 0.25, 0.5, 1, 0.15)	191.9	13.4
28	(40, 0.25, 0.5, 2, 0.05)	228.2	3.8
29	(40, 0.25, 0.5, 2, 0.15)	231.7	7.3
30	(40, 0.5, 0.25, 0.5, 0.05)	151.0	27.2
31	(40, 0.5, 0.25, 0.5, 0.15)	144.6	27.5
32	(40, 0.5, 0.25, 1, 0.05)	240.9	1.4
33	(40, 0.5, 0.25, 1, 0.15)	192.2	4.5
34	(40, 0.5, 0.25, 2, 0.05)	245.5	2.0
35	(40, 0.5, 0.25, 2, 0.15)	230.9	6.6

TABLE B.5: RegNet2 (paradigm: CLONALG, fitness: WC) hyperparameter investigation

	Hyperparameters	Avg Win Count	Avg L2 Loss
0	(10, 0.25, 0, 0.05, 0.15)	164.5	19.5
1	(10, 0.25, 0, 0.15, 0.05)	135.2	27.4
2	(10, 0.25, 1, 0.05, 0.15)	239.3	2.9
3	(10, 0.25, 1, 0.15, 0.05)	185.3	16.6
4	(10, 0.5, 0, 0.05, 0.15)	216.6	7.3
5	(10, 0.5, 0, 0.15, 0.05)	144.7	10.9
6	(10, 0.5, 1, 0.05, 0.15)	199.2	9.9
7	(10, 0.5, 1, 0.15, 0.05)	114.5	27.0
8	(20, 0.25, 0, 0.05, 0.15)	215.3	4.2
9	(20, 0.25, 0, 0.15, 0.05)	154.2	27.1
10	(20, 0.25, 1, 0.05, 0.15)	188.7	17.8
11	(20, 0.25, 1, 0.15, 0.05)	156.4	25.1
12	(20, 0.5, 0, 0.05, 0.15)	199.8	20.1
13	(20, 0.5, 0, 0.15, 0.05)	136.8	21.9
14	(20, 0.5, 1, 0.05, 0.15)	230.3	4.1
15	(20, 0.5, 1, 0.15, 0.05)	137.8	42.2
16	(40, 0.25, 0, 0.05, 0.15)	197.5	14.5
17	(40, 0.25, 0, 0.15, 0.05)	142.7	27.5
18	(40, 0.25, 1, 0.05, 0.15)	187.4	17.0
19	(40, 0.25, 1, 0.15, 0.05)	177.6	15.3
20	(40, 0.5, 0, 0.05, 0.15)	220.9	2.9
21	(40, 0.5, 0, 0.15, 0.05)	148.9	27.0
22	(40, 0.5, 1, 0.05, 0.15)	215.0	9.6
23	(40, 0.5, 1, 0.15, 0.05)	192.4	10.5

TABLE B.6: RegNet2 (paradigm: GA, fitness: WC) hyperparameter investigation

	Hyperparameters	Avg Win Count	Avg L2 Loss
0	(10, 0.25, 0.5, 0.5, 0.05)	28.9	327.8
1	(10, 0.25, 0.5, 0.5, 0.15)	15.8	594.1
2	(10, 0.25, 0.5, 1, 0.05)	28.5	132.5
3	(10, 0.25, 0.5, 1, 0.15)	36.8	100.9
4	(10, 0.25, 0.5, 2, 0.05)	20.7	694.0
5	(10, 0.25, 0.5, 2, 0.15)	24.9	730.5
6	(10, 0.5, 0.25, 0.5, 0.05)	23.0	382.0
7	(10, 0.5, 0.25, 0.5, 0.15)	29.0	375.6
8	(10, 0.5, 0.25, 1, 0.05)	105.7	53.2
9	(10, 0.5, 0.25, 1, 0.15)	36.6	111.2
10	(10, 0.5, 0.25, 2, 0.05)	38.5	274.3
11	(10, 0.5, 0.25, 2, 0.15)	17.0	298.2
12	(20, 0.25, 0.5, 0.5, 0.05)	21.7	272.0
13	(20, 0.25, 0.5, 0.5, 0.15)	28.3	273.6
14	(20, 0.25, 0.5, 1, 0.05)	39.8	53.6
15	(20, 0.25, 0.5, 1, 0.15)	86.6	33.2
16	(20, 0.25, 0.5, 2, 0.05)	44.3	287.1
17	(20, 0.25, 0.5, 2, 0.15)	30.0	198.4
18	(20, 0.5, 0.25, 0.5, 0.05)	12.9	290.5
19	(20, 0.5, 0.25, 0.5, 0.15)	19.6	303.3
20	(20, 0.5, 0.25, 1, 0.05)	45.0	22.0
21	(20, 0.5, 0.25, 1, 0.15)	59.0	25.8
22	(20, 0.5, 0.25, 2, 0.05)	47.4	132.3
23	(20, 0.5, 0.25, 2, 0.15)	18.7	256.6
24	(40, 0.25, 0.5, 0.5, 0.05)	33.1	135.7
25	(40, 0.25, 0.5, 0.5, 0.15)	26.3	209.8
26	(40, 0.25, 0.5, 1, 0.05)	74.4	9.8
27	(40, 0.25, 0.5, 1, 0.15)	81.0	6.2
28	(40, 0.25, 0.5, 2, 0.05)	41.9	69.0
29	(40, 0.25, 0.5, 2, 0.15)	50.0	70.1
30	(40, 0.5, 0.25, 0.5, 0.05)	31.4	176.8
31	(40, 0.5, 0.25, 0.5, 0.15)	30.4	138.8
32	(40, 0.5, 0.25, 1, 0.05)	88.4	6.2
33	(40, 0.5, 0.25, 1, 0.15)	100.2	10.9
34	(40, 0.5, 0.25, 2, 0.05)	43.2	44.1
35	(40, 0.5, 0.25, 2, 0.15)	112.4	13.9

TABLE B.7: Dict-opt (paradigm: CLONALG, fitness: $L2$) hyperparameter investigation

	Hyperparameters	Avg Win Count	Avg L2 Loss
0	(10, 0.25, 0, 0.05, 0.15)	18.2	505.0
1	(10, 0.25, 0, 0.15, 0.05)	23.1	824.4
2	(10, 0.25, 1, 0.05, 0.15)	12.6	456.4
3	(10, 0.25, 1, 0.15, 0.05)	25.4	718.8
4	(10, 0.5, 0, 0.05, 0.15)	26.7	385.9
5	(10, 0.5, 0, 0.15, 0.05)	19.4	365.9
6	(10, 0.5, 1, 0.05, 0.15)	7.8	638.5
7	(10, 0.5, 1, 0.15, 0.05)	16.1	661.8
8	(20, 0.25, 0, 0.05, 0.15)	24.4	261.4
9	(20, 0.25, 0, 0.15, 0.05)	22.0	739.9
10	(20, 0.25, 1, 0.05, 0.15)	36.6	127.2
11	(20, 0.25, 1, 0.15, 0.05)	17.9	831.1
12	(20, 0.5, 0, 0.05, 0.15)	26.4	465.3
13	(20, 0.5, 0, 0.15, 0.05)	26.8	1069.0
14	(20, 0.5, 1, 0.05, 0.15)	23.8	487.4
15	(20, 0.5, 1, 0.15, 0.05)	16.5	914.5
16	(40, 0.25, 0, 0.05, 0.15)	14.9	405.0
17	(40, 0.25, 0, 0.15, 0.05)	24.4	644.9
18	(40, 0.25, 1, 0.05, 0.15)	19.7	305.3
19	(40, 0.25, 1, 0.15, 0.05)	16.9	495.3
20	(40, 0.5, 0, 0.05, 0.15)	30.2	320.9
21	(40, 0.5, 0, 0.15, 0.05)	18.6	639.6
22	(40, 0.5, 1, 0.05, 0.15)	31.2	332.8
23	(40, 0.5, 1, 0.15, 0.05)	15.2	765.9

TABLE B.8: Dict-opt (paradigm: GA, fitness: $L2$) hyperparameter investigation

	Hyperparameters	Avg Win Count	Avg L2 Loss
0	(10, 0.25, 0.5, 0.5, 0.05)	125.6	3.4
1	(10, 0.25, 0.5, 0.5, 0.15)	77.3	9.0
2	(10, 0.25, 0.5, 1, 0.05)	204.5	4.1
3	(10, 0.25, 0.5, 1, 0.15)	167.0	2.6
4	(10, 0.25, 0.5, 2, 0.05)	131.8	31.6
5	(10, 0.25, 0.5, 2, 0.15)	133.2	30.8
6	(10, 0.5, 0.25, 0.5, 0.05)	91.2	4.3
7	(10, 0.5, 0.25, 0.5, 0.15)	234.2	2.2
8	(10, 0.5, 0.25, 1, 0.05)	233.0	1.9
9	(10, 0.5, 0.25, 1, 0.15)	185.8	2.2
10	(10, 0.5, 0.25, 2, 0.05)	45.3	30.7
11	(10, 0.5, 0.25, 2, 0.15)	133.7	31.4
12	(20, 0.25, 0.5, 0.5, 0.05)	122.6	4.5
13	(20, 0.25, 0.5, 0.5, 0.15)	234.0	2.9
14	(20, 0.25, 0.5, 1, 0.05)	259.5	2.0
15	(20, 0.25, 0.5, 1, 0.15)	251.6	2.0
16	(20, 0.25, 0.5, 2, 0.05)	135.2	31.6
17	(20, 0.25, 0.5, 2, 0.15)	134.9	30.7
18	(20, 0.5, 0.25, 0.5, 0.05)	174.6	2.6
19	(20, 0.5, 0.25, 0.5, 0.15)	198.4	2.6
20	(20, 0.5, 0.25, 1, 0.05)	256.8	2.0
21	(20, 0.5, 0.25, 1, 0.15)	255.6	1.9
22	(20, 0.5, 0.25, 2, 0.05)	133.6	30.3
23	(20, 0.5, 0.25, 2, 0.15)	223.2	2.1
24	(40, 0.25, 0.5, 0.5, 0.05)	102.1	3.1
25	(40, 0.25, 0.5, 0.5, 0.15)	150.2	3.3
26	(40, 0.25, 0.5, 1, 0.05)	251.1	2.0
27	(40, 0.25, 0.5, 1, 0.15)	255.1	2.0
28	(40, 0.25, 0.5, 2, 0.05)	254.5	2.1
29	(40, 0.25, 0.5, 2, 0.15)	261.8	2.2
30	(40, 0.5, 0.25, 0.5, 0.05)	259.1	1.9
31	(40, 0.5, 0.25, 0.5, 0.15)	163.4	2.4
32	(40, 0.5, 0.25, 1, 0.05)	259.4	1.9
33	(40, 0.5, 0.25, 1, 0.15)	253.5	2.0
34	(40, 0.5, 0.25, 2, 0.05)	258.7	1.9
35	(40, 0.5, 0.25, 2, 0.15)	260.1	1.9

TABLE B.9: RegNet1 (paradigm: CLONALG, fitness: $L2$) hyperparameter investigation

	Hyperparameters	Avg Win Count	Avg L2 Loss
0	(10, 0.25, 0, 0.05, 0.15)	183.5	2.5
1	(10, 0.25, 0, 0.15, 0.05)	129.4	4.0
2	(10, 0.25, 1, 0.05, 0.15)	92.8	3.2
3	(10, 0.25, 1, 0.15, 0.05)	96.5	8.3
4	(10, 0.5, 0, 0.05, 0.15)	97.5	4.3
5	(10, 0.5, 0, 0.15, 0.05)	111.2	3.8
6	(10, 0.5, 1, 0.05, 0.15)	206.4	2.4
7	(10, 0.5, 1, 0.15, 0.05)	73.4	6.6
8	(20, 0.25, 0, 0.05, 0.15)	145.5	2.5
9	(20, 0.25, 0, 0.15, 0.05)	59.0	5.6
10	(20, 0.25, 1, 0.05, 0.15)	207.2	2.2
11	(20, 0.25, 1, 0.15, 0.05)	143.4	2.4
12	(20, 0.5, 0, 0.05, 0.15)	235.2	2.6
13	(20, 0.5, 0, 0.15, 0.05)	140.5	4.1
14	(20, 0.5, 1, 0.05, 0.15)	141.3	3.0
15	(20, 0.5, 1, 0.15, 0.05)	131.8	3.5
16	(40, 0.25, 0, 0.05, 0.15)	234.9	2.4
17	(40, 0.25, 0, 0.15, 0.05)	90.5	8.4
18	(40, 0.25, 1, 0.05, 0.15)	151.7	2.4
19	(40, 0.25, 1, 0.15, 0.05)	216.5	3.7
20	(40, 0.5, 0, 0.05, 0.15)	150.4	2.4
21	(40, 0.5, 0, 0.15, 0.05)	142.8	3.4
22	(40, 0.5, 1, 0.05, 0.15)	91.2	3.3
23	(40, 0.5, 1, 0.15, 0.05)	191.1	4.8

TABLE B.10: RegNet1 (paradigm: GA, fitness: $L2$) hyperparameter investigation

	Hyperparameters	Avg Win Count	Avg L2 Loss
0	(10, 0.25, 0.5, 0.5, 0.05)	113.7	27.3
1	(10, 0.25, 0.5, 0.5, 0.15)	48.6	23.1
2	(10, 0.25, 0.5, 1, 0.05)	113.2	27.0
3	(10, 0.25, 0.5, 1, 0.15)	50.0	9.8
4	(10, 0.25, 0.5, 2, 0.05)	66.3	27.4
5	(10, 0.25, 0.5, 2, 0.15)	167.8	20.0
6	(10, 0.5, 0.25, 0.5, 0.05)	109.3	4.8
7	(10, 0.5, 0.25, 0.5, 0.15)	47.5	22.4
8	(10, 0.5, 0.25, 1, 0.05)	170.7	3.0
9	(10, 0.5, 0.25, 1, 0.15)	154.7	4.4
10	(10, 0.5, 0.25, 2, 0.05)	108.0	32.3
11	(10, 0.5, 0.25, 2, 0.15)	70.6	14.9
12	(20, 0.25, 0.5, 0.5, 0.05)	126.0	7.8
13	(20, 0.25, 0.5, 0.5, 0.15)	74.3	8.0
14	(20, 0.25, 0.5, 1, 0.05)	163.5	4.5
15	(20, 0.25, 0.5, 1, 0.15)	157.8	2.4
16	(20, 0.25, 0.5, 2, 0.05)	66.8	27.5
17	(20, 0.25, 0.5, 2, 0.15)	112.3	27.1
18	(20, 0.5, 0.25, 0.5, 0.05)	154.5	7.7
19	(20, 0.5, 0.25, 0.5, 0.15)	103.4	6.8
20	(20, 0.5, 0.25, 1, 0.05)	226.1	1.1
21	(20, 0.5, 0.25, 1, 0.15)	231.4	1.1
22	(20, 0.5, 0.25, 2, 0.05)	112.5	26.8
23	(20, 0.5, 0.25, 2, 0.15)	207.0	2.1
24	(40, 0.25, 0.5, 0.5, 0.05)	83.5	4.5
25	(40, 0.25, 0.5, 0.5, 0.15)	117.5	3.6
26	(40, 0.25, 0.5, 1, 0.05)	235.1	1.2
27	(40, 0.25, 0.5, 1, 0.15)	187.0	2.6
28	(40, 0.25, 0.5, 2, 0.05)	231.3	1.2
29	(40, 0.25, 0.5, 2, 0.15)	214.2	2.9
30	(40, 0.5, 0.25, 0.5, 0.05)	90.0	4.2
31	(40, 0.5, 0.25, 0.5, 0.15)	174.2	2.7
32	(40, 0.5, 0.25, 1, 0.05)	191.8	1.7
33	(40, 0.5, 0.25, 1, 0.15)	250.6	1.0
34	(40, 0.5, 0.25, 2, 0.05)	113.6	27.5
35	(40, 0.5, 0.25, 2, 0.15)	113.3	27.7

TABLE B.11: RegNet2 (paradigm: CLONALG, fitness: $L2$) hyperparameter investigation

	Hyperparameters	Avg Win Count	Avg L2 Loss
0	(10, 0.25, 0, 0.05, 0.15)	122.5	2.8
1	(10, 0.25, 0, 0.15, 0.05)	100.3	15.5
2	(10, 0.25, 1, 0.05, 0.15)	89.3	8.7
3	(10, 0.25, 1, 0.15, 0.05)	63.0	9.6
4	(10, 0.5, 0, 0.05, 0.15)	77.5	16.9
5	(10, 0.5, 0, 0.15, 0.05)	103.8	10.2
6	(10, 0.5, 1, 0.05, 0.15)	62.2	6.6
7	(10, 0.5, 1, 0.15, 0.05)	80.4	13.0
8	(20, 0.25, 0, 0.05, 0.15)	83.2	4.4
9	(20, 0.25, 0, 0.15, 0.05)	109.2	5.4
10	(20, 0.25, 1, 0.05, 0.15)	178.7	1.7
11	(20, 0.25, 1, 0.15, 0.05)	106.5	14.3
12	(20, 0.5, 0, 0.05, 0.15)	168.2	14.5
13	(20, 0.5, 0, 0.15, 0.05)	122.9	8.9
14	(20, 0.5, 1, 0.05, 0.15)	74.0	7.8
15	(20, 0.5, 1, 0.15, 0.05)	77.0	10.6
16	(40, 0.25, 0, 0.05, 0.15)	68.4	11.1
17	(40, 0.25, 0, 0.15, 0.05)	57.4	8.7
18	(40, 0.25, 1, 0.05, 0.15)	169.6	2.0
19	(40, 0.25, 1, 0.15, 0.05)	40.2	14.9
20	(40, 0.5, 0, 0.05, 0.15)	64.5	5.2
21	(40, 0.5, 0, 0.15, 0.05)	97.9	3.1
22	(40, 0.5, 1, 0.05, 0.15)	193.1	2.2
23	(40, 0.5, 1, 0.15, 0.05)	41.8	13.7

TABLE B.12: RegNet2 (paradigm: GA, fitness: $L2$) hyperparameter investigation

Bibliography

Oscar M Alonso, Fernando Nino, and Marcos Velez. A robust immune based approach to the iterated prisoner's dilemma. *Lecture notes in computer science*, pages 290–301, 2004. doi: 10.1007/978-3-540-30220-9_24. URL https://doi.org/10.1007/978-3-540-30220-9_24.

Robert Axelrod. The evolution of strategies in the iterated prisoner's dilemma. *Genetic Algorithms and Simulated Annealing*, 89:32–41, 1987. URL https://users.auth.gr/users/7/8/007287/public_html/Research/GameTheory/05PapersAdvanced/PrisonersDilemma/Evolving.pdf.

Kaushik Basu. The traveler's dilemma: Paradoxes of rationality in game theory. *The American Economic Review*, 84(2):391–395, 1994. ISSN 0002-8282. URL https://www.jstor.org/stable/2117865.

André Brandão, Pedro Pires, and Petia Georgieva. Reinforcement learning and neuroevolution in flappy bird game. In Aythami Morales, Julian Fierrez, José Salvador Sánchez, and Bernardete Ribeiro, editors, *Pattern Recognition and Image Analysis*, pages 225–236, Cham, 2019. Springer International Publishing. ISBN 978-3-030-31332-6. doi: 10.1007/978-3-030-31332-6_20. URL https://doi.org/10.1007/978-3-030-31332-6_20.

Jason Brownlee. *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee, 2011. ISBN 9781446785065. URL https://raw.githubusercontent.com/clever-algorithms/CleverAlgorithms/master/release/clever_algorithms.pdf.

C. A. Coello and N. Cruz Cortes. A parallel implementation of an artificial immune system to handle constraints in genetic algorithms: preliminary results. IEEE, 2002. doi: 10.1109/cec.2002.1007031. URL https://dx.doi.org/10.1109/cec.2002.1007031.

Andrew Connolly. Deep learning: Classifying astronomical images, 2020. URL https://www.astroml.org/astroML-notebooks/chapter9/astroml_chapter9_Deep_Learning_Classifying_Astronomical_Images.html.

George V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989. URL https://api.semanticscholar.org/CorpusID:3958369.

Charles Darwin. *On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life*, volume 11859. London: John Murray, 1831.

L. N. De Castro and J. Timmis. An artificial immune network for multimodal function optimization. IEEE, 2002. doi: 10.1109/cec.2002.1007011. URL https://dx.doi.org/10.1109/cec.2002.1007011.

L. N. De Castro and F. J. Von Zuben. Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6(3):239–251, 2002. ISSN 1089-778X. doi: 10.1109/tevc.2002.1011539. URL https://dx.doi.org/10.1109/tevc.2002.1011539.

Leandro Nunes de Castro and Fernando J Von Zuben. ainet: an artificial immune network for data analysis. In *Data mining: a heuristic approach*, pages 231–260. IGI Global, 2002. URL https://www.dca.fee.unicamp.br/~vonzuben/research/lnunes_dout/artigos/DMHA.PDF.

Stephane Doncieux, Nicolas Bredeche, Jean-Baptiste Mouret, and Agoston E Eiben. Evolutionary robotics: what, why, and where to. *Frontiers in Robotics and AI*, 2:4, 2015. doi: 10.3389/frobt.2015.00004. URL https://doi.org/10.3389/frobt.2015.00004.

Alex Fabrikant, Christos Papadimitriou, and Kunal Talwar. The complexity of pure nash equilibria. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 604–612, 2004. doi: 10.1145/1007352.1007445. URL https://doi.org/10.1145/1007352.1007445.

Michael P. Fay and Michael A. Proschan. Wilcoxon-mann-whitney or t-test? on assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics surveys*, 4:1–39, 2010. URL https://api.semanticscholar.org/CorpusID:8209167.

S. Forrest, A.S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *Proceedings of 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 202–212, 1994. doi: 10.1109/RISP.1994.296580. URL https://doi.org/10.1109/RISP.1994.296580.

Robert R. Gibbons. A primer in game theory. 1992. URL https://api.semanticscholar.org/CorpusID:10248389.

Jennifer Golbeck. Evolving strategies for the prisoner's dilemma. *Advances in Intelligent Systems, Fuzzy Systems, and Evolutionary Computation*, 2002:299, 2002. URL https://www.cs.umd.edu/users/golbeck/downloads/JGolbeck_prison.pdf.

Julie Greensmith, Uwe Aickelin, and Steve Cayzer. Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection. In *Artificial Immune Systems: 4th International Conference, ICARIS 2005, Banff, Alberta, Canada, August 14-17, 2005. Proceedings 4*, pages 153–167. Springer, 2005. doi: 10.1007/11536444_12. URL https://doi.org/10.1007/11536444_12.

Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016. doi: 10.48550/arXiv.1604.00772. URL https://doi.org/10.48550/arXiv.1604.00772.

Ahmad Hassanat, Khalid Almohammadi, Esra'a Alkafaween, Eman Abunawas, Awni Mansoar Hammouri, and V. B. Surya Prasath. Choosing mutation and crossover ratios for genetic algorithms - a review with a new dynamic approach. *Inf.*, 10:390, 2019. URL https://api.semanticscholar.org/CorpusID:209353552.

Sabine Hauert, Jean-Christophe Zufferey, and Dario Floreano. Evolved swarming without positioning information: an application in aerial communication relay. *Autonomous Robots*, 26:21–32, 2009. URL https://api.semanticscholar.org/CorpusID:5533715.

Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366, 2014. doi: 10.1109/TCIAIG.2013.2294713. URL https://doi.org/10.1109/TCIAIG.2013.2294713.

Yulan He, Siu Cheung Hui, and Edmund Ming-Kit Lai. Automatic timetabling using artificial immune system. In *Algorithmic Applications in Management: First International Conference, AAIM 2005, Xian, China, June 22-25, 2005. Proceedings 1*, pages 55–65. Springer, 2005. doi: 10.1007/11496199_8. URL https://doi.org/10.1007/11496199_8.

Teck-Hua Ho, Colin Camerer, and Keith Weigelt. Iterated dominance and iterated best response in experimental" p-beauty contests". *The American Economic Review*, 88 (4):947–969, 1998. ISSN 0002-8282. URL https://www.jstor.org/stable/117013.

Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, volume 2, pages 985–990 vol.2, 2004. doi: 10.1109/IJCNN.2004.1380068. URL https://doi.org/10.1109/IJCNN.2004.1380068.

Jerne. Towards a network theory of the immune system. *Annales De L'institut Pasteur. Immunologie*, 125:373–389, 1974.

Kim Jungwon and P. J. Bentley. Towards an artificial immune system for network intrusion detection: an investigation of dynamic clonal selection. IEEE, 2002. doi: 10.1109/cec.2002.1004382. URL https://dx.doi.org/10.1109/cec.2002.1004382.

John Maynard Keynes. *The general theory of employment, interest and money*. London : Macmillan, 1936., 1936. URL https://search.library.wisc.edu/catalog/999623618402121.

Oliver Kramer. Genetic algorithm essentials. *Studies in Computational Intelligence*, 2017. doi: 10.1007/978-3-319-52156-5. URL https://doi.org/10.1007/978-3-319-52156-5.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Alex Lenail. Nn-svg, 2018. URL https://alexlenail.me/NN-SVG/index.html.

Jun-qing Li, Zheng-min Liu, Chengdong Li, and Zhi-xin Zheng. Improved artificial immune system algorithm for type-2 fuzzy flexible job shop scheduling problem. *IEEE Transactions on Fuzzy Systems*, 29(11):3234–3248, 2021. doi: 10.1109/TFUZZ.2020.3016225. URL https://doi.org/10.1109/TFUZZ.2020.3016225.

Jason D. Lohn, Derek S. Linden, and Gregory Hornby. Advanced antenna design for a nasa small satellite mission. 2008. URL https://api.semanticscholar.org/CorpusID:39881510.

Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. URL https://cs.gmu.edu/~sean/book/metaheuristics/Essentials.pdf.

Patrick MacAlpine and Peter Stone. Overlapping layered learning. *Artificial Intelligence*, 254:21–43, 2018. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2017.09.001. URL https://www.sciencedirect.com/science/article/pii/S0004370217301066.

K.F. Man, K.S. Tang, and S. Kwong. Genetic algorithms: concepts and applications [in engineering design]. *IEEE Transactions on Industrial Electronics*, 43(5):519–534, 1996. doi: 10.1109/41.538609. URL https://doi.org/10.1109/41.538609.

Henry B. Mann and Douglas R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18:50–60, 1947. URL https://api.semanticscholar.org/CorpusID:14328772.

Risto Miikkulainen. *Neuroevolution*, pages 716–720. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_589. URL https://doi.org/10.1007/978-0-387-30164-8_589.

Francesco Mondada and Dario Floreano. *Evolution and mobile autonomous robotics*, pages 221–249. Springer, 2005. URL https://link.springer.com/chapter/10.1007/3-540-61093-6_10.

Herve Moulin. *Game theory for the social sciences*. NYU press, 1986. ISBN 0814764231. URL https://books.google.co.jp/books?hl=fr&lr=&id=96ewJBssYKEC&oi=fnd&pg=PR3&dq=herve+moulin+game+theory+for+the+social+sciences&ots=K8it8b8zWK&sig=9BA2DRYClWYpO4ffSxjHItIpUXg.

Rosemarie Nagel. Unraveling in guessing games: An experimental study. *The American economic review*, 85(5):1313–1326, 1995. ISSN 0002-8282. URL https://www.jstor.org/stable/2950991.

Marcin L. Pilat and Franz Oppacher. *Evolution of Khepera Robotic Controllers with Hierarchical Genetic Programming Techniques*, pages 43–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-32364-8. doi: 10.1007/3-540-32364-3_3. URL https://doi.org/10.1007/3-540-32364-3_3.

Sebastian Risi and Julian Togelius. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 9: 25–41, 2014. URL https://api.semanticscholar.org/CorpusID:11245845.

Poonam Savsani, Ramdevsinh L Jhala, and Vimal Savsani. Effect of hybridizing biogeography-based optimization (bbo) technique with artificial immune algorithm (aia) and ant colony optimization (aco). *Applied Soft Computing*, 21:542–553, 2014. ISSN 1568-4946. doi: 10.1016/j.asoc.2014.03.011. URL https://doi.org/10.1016/j.asoc.2014.03.011.

M. Srinivas and L.M. Patnaik. Genetic algorithms: a survey. *Computer*, 27(6):17–26, 1994. doi: 10.1109/2.294849. URL https://doi.org/10.1109/2.294849.

Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002. doi: 10.1162/106365602320169811. URL https://ieeexplore.ieee.org/document/6790655.

Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life*, 15(2):185–212, 04 2009. ISSN 1064-5462. doi: 10.1162/artl.2009.15.2.15202. URL https://doi.org/10.1162/artl.2009.15.2.15202.

K.O. Stanley, B.D. Bryant, and R. Miikkulainen. Real-time neuroevolution in the nero video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668, 2005. doi: 10.1109/TEVC.2005.856210. URL https://doi.org/10.1109/TEVC.2005.856210.

J. Timmis, M. Neal, and J. Hunt. Data analysis using artificial immune systems, cluster analysis and kohonen networks: some comparisons. In *IEEE SMC'99 Conference*

Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028), volume 3, pages 922–927 vol.3, 1999. doi: 10.1109/ICSMC.1999.823351. URL https://doi.org/10.1109/ICSMC.1999.823351.

Albert W Tucker and Philip D Straffin Jr. The mathematics of tucker: A sampler. *The Two-Year College Mathematics Journal*, 14(3):228–232, 1983. doi: 10.2307/3027092. URL https://doi.org/10.2307/3027092.

Ezgi Deniz Ulker and Sadik Ulker. Comparison study for clonal selection algorithm and genetic algorithm. *ArXiv*, abs/1209.2717, 2012. URL https://api.semanticscholar.org/CorpusID:6656084.

Aymeric Vié. Evolutionary strategies with analogy partitions in p-guessing games. *SSRN Electronic Journal*, 2020. ISSN 1556-5068. doi: 10.2139/ssrn.3645059. URL https://dx.doi.org/10.2139/ssrn.3645059.

Claus Wedekind and Manfred Milinski. Human cooperation in the simultaneous and the alternating prisoner's dilemma: Pavlov versus generous tit-for-tat. *Proceedings of the National Academy of Sciences*, 93(7):2686–2689, 1996. doi: 10.1073/pnas.93.7.2686. URL https://doi.org/10.1073/pnas.93.7.2686.

P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi: 10.1109/5.58337. URL https://doi.org/10.1109/5.58337.

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:196–202, 1945. URL https://api.semanticscholar.org/CorpusID:53662922.