computation

MDPI

Article

An Evolutionary Computing Model for the Study of Within-Host Evolution

Antonio Gómez-Mompeán and Rafael Lahoz-Beltra *

Department of Biodiversity, Ecology and Evolution (Biomathematics), Faculty of Biological Sciences c/ Jose Antonio Novais 2, Complutense University of Madrid, 28040 Madrid, Spain; antogo12@ucm.es

* Correspondence: lahozraf@ucm.es

Received: 4 September 2019; Accepted: 6 January 2020; Published: 9 January 2020

Abstract: Evolution of an individual within another individual is known as within-host dynamics (WHD). The most common modeling technique to study WHD involves ordinary differential equations (ODEs). In the field of biology, models of this kind assume, for example, that both the number of viruses and the number of mouse cells susceptible to being infected change according to their interaction as stated in the ODE model. However, viruses can undergo mutations and, consequently, evolve inside the mouse, whereas the mouse, in turn, displays evolutionary mechanisms through its immune system (e.g., clonal selection), defending against the invading virus. In this work, as the main novelty, we propose an evolutionary WHD model simulating the coexistence of an evolving invader within a host. In addition, instead of using ODEs we developed an alternative methodology consisting of the hybridization of a genetic algorithm with an artificial immune system. Aside from the model, interest in biology, and its potential clinical use, the proposed WHD model may be useful in those cases where the invader exhibits evolutionary changes, for instance, in the design of anti-virus software, intrusion detection algorithms in a corporation's computer systems, etc. The model successfully simulates two intruder detection paradigms (i.e., humoral detection, danger detection) in which the intruder represents an evolving invader or guest (e.g., virus, computer program,) that infects a host (e.g., mouse, computer memory). The obtained results open up the possibility of simulating environments in which two entities (guest versus host) compete evolutionarily with each other when occupying the same space (e.g., organ cells, computer memory, network).

Keywords: within-host evolution; artificial immune system; hybridization of a genetic algorithm; evolutionary model; anti-virus software; intrusion detection algorithms

1. Introduction

Evolution is one of the central paradigms of contemporary biology. At present, there is no mechanism or biological phenomenon which can be explained outside of evolutionary thinking. Moreover, Darwin's theory of evolution has had a deep impact on other disciplines such as computer science. The possibility of simulating Darwinian evolution on a computer has resulted in an entire family of optimization algorithms (e.g., genetic algorithms (GAs)) grouped under the common umbrella of evolutionary computation [1]. Genetic algorithms are mainly oriented to solve optimization problems in economics, mathematics, geology, industry, social sciences, bioinformatics, etc. In some instances, GAs are also applied in biology to simulate evolution, for instance, the early stages of the evolution of the vertebrate eye [2] or the evolution of enzymes [3], i.e., proteins with catalytic function within cells or the pattern of stripes on the skin of zebras [4] are successfully simulated with a genetic algorithm. However, in all these examples, the evolution of a population of individuals occurs in a "non-living environment" composed of water, light, air, etc. Of course, the organisms living in an environment modify their physical area. Within this framework, the selection

of organisms that will give rise to a new generation of individuals rests on their fitness value. The fitness value confers to an individual a probability of selection based on the quality of the solution for which an individual stand. Individuals therefore represent solutions to a common problem: individuals have to conform to a non-living environment and, in particular, to one of the many environments on planet Earth. Consequently, and assuming some level of abstraction, it is possible to simulate biological evolution with a genetic algorithm.

However, what happens when the evolution of a population takes place in a "living environment?" As an example, a virus infects an animal, which is referred to as host, evolving into the host animal. In this scenario, the viruses evolve within the host, but also the animal holds evolutionary mechanisms directed to defend the host from the harmful virus. In the host, the evolutionary mechanisms involve the immune system. The immune system represents the organism's defense against infectious organisms (e.g., virus, bacteria) through a sequence of steps known as immune response. Currently, in a similar way as GAs are inspired by organic evolution (i.e., Darwinian evolution), the artificial immune system (AIS) algorithms [5–10] are inspired by the immune system of vertebrates. These algorithms share some of the characteristics of the GAs, and they are currently used for solving practical problems [11,12].

In the field of biology, the coexistence of an individual, for example a virus, within another individual, for example a mouse, is known as "within-host dynamics" (WHD) [13,14]. In this work, we refer to the invader individual (e.g., virus, bacteria) as "guest", setting the term "host" to the recipient individual (e.g., mouse). In this paper, we designed an evolutionary WHD simulation model in which the guest evolves inside a host. Consequently, a novel ingredient of the WHD model is that it includes guest evolution. Evolution of the guest is simulated with a GA, whereas the defense mechanisms of the host are simulated through AIS algorithms. Another original feature of the model lies in its methodology showing how the use of heuristic algorithms inspired by biology is an alternative to classical mathematical modeling. We believe there is an interest in studying how a guest evolves within a host which defends its own integrity through the immune system. Moreover, we think that the interest of this issue goes beyond biology and may be relevant in other fields, e.g., computer science.

The use of ordinary differential equations (ODEs) has, to date, been one of the most appropriate techniques for understanding the mechanisms that govern host–guest kinetics [15], assuming, like in this paper, a pathogen guest. In the realm of quasi-species theory [16] (i.e., a theory about Darwinian evolution in self-replicating organisms), ODEs have also been successfully applied, e.g., in the study of HIV (Human Immunodeficiency Virus) virus evolution [17]. Nevertheless, the WHD term generally refers to a host–guest coexistence model in which guest evolution is not included. In the scope of the present work, other modeling techniques have been applied. For example in Reference [18], their authors conceive a qualitative simulation model of a λ phage (a virus, i.e. guest) that infects *Escherichia coli* bacterium (i.e. host).

According to the model described in Reference [15], the within-host dynamics can be modeled as an ODE system:

$$\begin{aligned} \frac{dT}{dt} &= -\beta V T \\ \frac{dV}{dt} &= r \beta V T - \gamma V \end{aligned} \tag{1}$$

where T is the number of healthy host cells susceptible to infection, β is the rate of infection, V is the number of viruses, r is a model parameter related with viral replication, and γ is the mortality rate of infected cells.

In 2005, a paper [19] on the simulated evolution of HIV within an infected host was published, studying the effects of mutations and recombination (crossover). However, the model was a GA which does not include the host simulation. The authors only studied the effects of a simulated stochastic processes on a living host, e.g., in an individual affected by the disease.

In the present work, the evolutionary WHD simulation model was inspired by a specific biological scenario and then applied to an elementary example in computer science. In the biological scenario in which we were inspired by, a virus (guest) infects and evolves within the organ of a mouse (host) which defends its integrity from infection. Afterwards, we applied the model in computer science, considering a hypothetical example in which a computer program that is initially helpful to a computer evolves into a harmful program. Therefore, no matter which scenario is considered in the simulation experiments, two algorithms that perform opposite roles are confronted: the GA evolves the guest within the host, while a host artificial immune system (AIS) algorithm defends the latter from the evolutionary proliferation of the guest. The study, modeling, and simulation of this situation is the main novelty of this work. The WHD model is based on GAs and AIS, because such algorithms are inspired by biology, and both are quite accurate in emulating the mechanisms present in the organisms. In addition, such algorithms are sufficiently studied and have numerous applications in computer science.

In Section 2 of this paper, we present the WHD model description and software realization including several simulation experiments of virus (guest) evolution within a mouse (host) with immune system. Section 2.2. explains the modeling and simulation of virus evolution, and Section 2.3. is a review of the AIS algorithms we applied in this paper. Following, Section 2.4. describes the architecture of the evolutionary WHD model—how the GAs and the AIS algorithms interact with each other, modeling the effect of the AIS on the evolution of viruses. Next, in Section 3, we explore the application of the model in the design of a program that detects the evolution of a malicious program in the memory of a hypothetical computer. Finally, Section 4 presents the entire results of the computer simulation experiments, and Section 5 discusses the possible impact of this work together with the general limitations of the model, suggesting future directions of advancement.

2. An Evolutionary WHD Model Inspired by Virus Evolution inside a Mouse with Immune System

In order to find the inspiration to design the evolutionary WHD model, we resorted to the following biological scenario. We simulated the evolution of an elementary model of virus using a GA, whereas the simulation of a mouse immune system was carried out by means of AIS algorithms. We assumed that virus evolution takes place within an infected cell belonging to a mouse organ. Virus evolution experiments were conducted both in the absence and under the effects of the immune system (Figure 1). The WHD model comprised a host (i.e., mouse) and a guest (i.e., virus) modelled as explained in the following section.

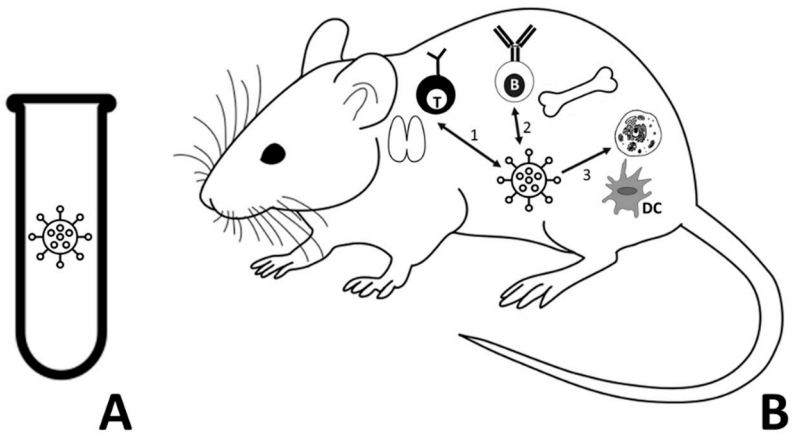

Figure 1. Evolution of the virus in two different environments. (**A**) The evolution of the virus within a non-living environment (e.g., in silico) is simulated via genetic algorithm (GA). (**B**) In a living environment (e.g., a mouse), the evolution of the virus is compromised by the presence of an immune

system. The immune system is simulated with artificial immune system (AIS) algorithms: (1) in the mouse thymus, T-lymphocytes have the ability to distinguish an intruder (e.g., virus) from any other organ of the mouse. Once the virus enters into the mouse, (2) the B-lymphocytes produced by the bone marrow are able to *detect* the virus and *neutralize* it, whereas (3) the dendritic cells (DCs) detect not the invader, but the *damage* caused by the virus in the mouse cells.

2.1. Host and Guest Modeling

The present model symbolically defines both the mouse (host) and the virus (guest) by means of a binary string:

$$\left\langle x_0, x_1, \ldots, x_l, x_{l+1}, x_{l+2}, ..., x_m, x_{m+1}, x_{m+2}, ..., x_n, x_{n+1}, x_{n+2}, ..., x_o \right\rangle$$

with $x \in (0,1)$. The binary string is organized in four substrings, reserving the first three to the definition of the identity of the mouse or virus and the fourth to the genome. Therefore, in both the mouse and the virus, the first three substrings are the sequences reserved for the AIS algorithms, and once defined they cannot be modified. In the case of the virus, only the fourth sequence is assigned to the GA, being subjected to evolutionary changes, e.g., by mutation.

The first substring $\left\langle x_0, x_1, \ldots, x_l \right\rangle$ is a label that allows the identification of the mouse or virus. A second substring $\left\langle x_{l+1}, x_{l+2}, ..., x_m \right\rangle$ represents in the mouse the organ in which the infection occurs and in the virus the protein envelope surrounding its genome, i.e., RNA (Ribonucleic acid, is the viral genetic material). The third substring $\left\langle x_{m+1}, x_{m+2}, ..., x_n \right\rangle$ is a recognition sequence for the immune system. In the biological realm, the third sequence would be simulating the epitope or antigenic determinant. This means that it is a binary sequence performing the role of identifier, which is recognized by antibodies (B-lymphocytes, Figure 1), the histocompatibility antigens (T-lymphocytes, Figure 1), etc. Finally, a fourth substring $\left\langle x_{n+1}, x_{n+2}, ..., x_o \right\rangle$ models the genome, i.e., the DNA (Deoxyribonucleic acid, is the mouse genetic material) from the mouse cells or the viral RNA.

In the mouse model, we included 12 possible organs. In the simulation experiments, a random organ is chosen to be the one infected by the virus. Inside this organ, the evolution of the virus will be simulated with the GA. Each organ of the mouse has a specific binary string with some substrings common to all organs and other variables:

$$\left\langle 0,0,0,0 \right\rangle \left\langle x_{l+1}, x_{l+2}, ..., x_m \right\rangle \left\langle 0,1,0,1,0,1,0,1,0,1, ..., x_n \right\rangle$$

The string above shows the common substring $\left\langle 0,0,0,0 \right\rangle$ that identifies the mouse (4 bits), the variable substring $\left\langle x_{l+1}, x_{l+2}, ..., x_m \right\rangle$ labeling the infected organ (4 bits), and the epitope string $\left\langle 0,1,0,1,0,1,0,1,0,1, ..., x_n \right\rangle$ which includes a common part and another variable (16 bits). Although the mouse genome is not used in simulation experiments, as it does not undergo evolutionary changes, for formal reasons, it has been included in the model (64 bits):

$$\langle 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\rangle$$

In this paper, we simulated an RNA virus. In the virus model, the first three substrings (virus identification, protein envelope, epitope) were set to the values shown below:

$$\left\langle 0,1,0,0 \right\rangle \left\langle 0,0,1,0 \right\rangle \left\langle 1,1,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,1,1,0,1,1,1,1 \right\rangle$$

The genome of the virus (i.e., RNA) was set-up as a zero sequence except for three 4 bit substrings representing the three genes of interest (64 bits):

$$\langle 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0\rangle$$

In the present WHD model, for both mouse and virus, the first three strings with a fixed setting play an immunological role and, therefore, do not change, e.g., through mutations. Thus, these strings are not subject to evolution. Consequently, the evolution of the virus via GA within the mouse will take place because of GA operations on the viral genome or RNA whose string is shown above.

2.2. An Evolutionary Algorithm for Viral Cycle Simulation

Genetic algorithms are stochastic optimization methods inspired by the principles of natural selection [1] established by Darwin's evolutionary theory in 1859. At present, they are used in optimization and search problems, being trial–error methods in which the space of solutions is continuously explored and evaluated until an optimum is found (global or local). A simple GA comprises the following steps. First, (a) a set of solutions (individuals) is randomly generated. Then, (b) the goodness or validity of the solutions (adaptation to the environment) is evaluated, i.e., their fitness. Those solutions with a higher fitness value will have a greater probability of surviving and, therefore, of reproducing, i.e., of surviving to the next generation (gene propagation). The solutions that go to the next generation will be modified because of genetic mechanisms, in particular, mutations and/or genetic crossover which are simulated in GAs. The search process is repeated until an optimal solution is found.

The RNA virus model was inspired by the HIV virus simulated in Reference [19], and we modeled it with the binary string described in Section 2.1. Accordingly, and to simplify the model, many of the biological features of the virus [20–25] were not included in the present paper.

Figure 2 shows a simple GA adapted to the evolutionary dynamics of a virus. The model assumes that once a virus enters (E) inside a host cell, evolution occurs within the cell, in the absence of the release step (R). Initially (t = 0), we generated a population of random virions, that is, a population of binary substrings (64 bits) modeling the viral RNA. Note that, as indicated in the previous section, the first three substrings with immunological significance were set in advance with arbitrary values. These lists were common to all viruses and were not subject to any evolutionary change. Next, we simulated the replication (R) and selection (S) of the virions that would pass to the next generation. The RS step involved the evaluation of the fitness of each viral particle. Once the virus population was assessed, the selection was carried out by the wheel parent selection method [1,3]. According to this method, all virions of the population become parental individuals. The probability of being selected depends on their fitness. In a roulette wheel, we define as many slots as virions have population, playing each virion to a slot whose surface is proportional to its fitness. The roulette wheel spin is simulated by calculating a probability value $p(i)$, where i is the RNA string number or its position in the viral population:

$$p(i) = \frac{F_i}{\sum_i F_i} \tag{2}$$

where F_i is the fitness of a viral particle i and $\sum_i F_i$ the total fitness of the population. Finally, we get a random number U_i being $U_i \in [0,1]$. The first virion or RNA string whose $p(i)$ is greater than or equal to the random number U_i will be reproduced once passing to the next generation. The procedure is repeated until a complete population of virions is obtained, maintaining a constant population size.

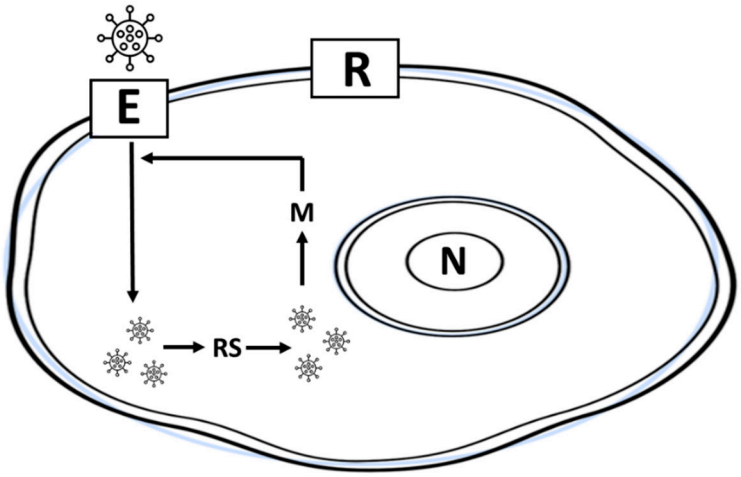

Figure 2. Evolutionary cycle of the virus inside a mouse cell. Once a virus enters into the cell (E), its evolution will take place assuming that offspring (i.e., virions) are not released to the outside (R). Each generation of virions is the result of a replication-selection step (RS), the mutation (M) being the genetic mechanism responsible for variability in the viral population. Although not included in this model, (N) refers the cell nucleus where the genome of the infected cell is located.

The fitness of the virions is calculated according to the criterion introduced in Reference [19]. In the simulation experiments, we defined a target genome or RNA including three genes or binary strings with sequence values: G_1 = {1,1,0,0}, G_2 = {1,0,1,0}, and G_3 = {1,0,0,1}. Thus, we assumed that the rest of the RNA string could hold any sequence, being the sequences of the three genes those that significantly contributed to virus fitness and, therefore, to viral evolution. Following, for each virion i and for the three regarded genes, the Hamming distance D_H was calculated between the RNA sequence of the virion w_j^v and the RNA target sequence w_j^t:

$$D_H = \sum_j \left| w_j^v - w_j^t \right| \tag{3}$$

Applying the criterion [19], the fitness value of virion i is given by the number of genes whose sequence matches the RNA target sequence. That is, the fitness F_i will be 0.33, 0.67 or 1.00 if one of the three genes, any two genes or the three genes simultaneously exhibit RNA target sequences:

$$F_i = \begin{cases} G_1 \vee G_2 \vee G_3 & , \quad 0.33\,w \\ (G_1 \wedge G_2) \vee (G_1 \wedge G_3) \vee (G_2 \wedge G_3) & , \quad 0.67w \\ G_1 \wedge G_2 \wedge G_3 & , \quad 1.00\,w \end{cases} \tag{4}$$

Note that the above scores are multiplied by a weight (w) value. Once the next generation of virions is obtained, the mutation (M) is applied, being simulated by the flip a bit technique [1,3]. In the GA, we did not include recombination or crossover. Since viruses are not released to the exterior (R) of the cell, the evolutionary loop is repeated until some convergence criterion is reached. The evolution of the viral population is confined inside the cell of a mouse organ, being evaluated its evolution by means of a performance graph (average fitness per generation).

2.3. Mouse Infection Model

The host subject of the modeling is a mouse, representing it symbolically according to the binary string defined in Section 2.1. The mouse has an immune system for which a simulation is carried out via a family of algorithms grouped under the common term of AIS [7–9]. This immune system provides the mouse with the ability to detect any external guest, e.g., virus. In immunology, virus detection is accomplished according two possible paradigms: either through intruder recognition (humoral immune response) or by the disturbances that intruder causes within the mouse (theory of

danger). Next, we review the AIS algorithms that we have implemented and customized to the mouse model.

2.3.1. Thymus Model and Simulation: Negative Selection Algorithm

The algorithm simulates the function performed by the thymus, a small gland located between the lungs. Lymphocytes (i.e., cells responsible for fighting against external infectious intruders such as virus, bacteria, and including cancer cells) travel through the bone marrow (i.e., a tissue that fills the bones) to the thymus. Once there, the lymphocytes become mature T-cells which are capable of binding to external intruders, the latter being referred to as antigens (Ag). Antigens have a bond site termed an epitope. A particular class of lymphocytes known as B-cells secrete proteins called immunoglobulins or antibodies (Ab). These antibodies also have a binding site called a paratope. The function of antibodies is to recognize the antigens, binding to them in an antigen–antibody (Ag–Ab) reaction. As a result of this reaction, the antigen (i.e., the virus) is neutralized. In the present model, we simulated the Ag epitope with the $\langle x_{m+1}, x_{m+2}, \ldots, x_n \rangle$ substring and the Ab paratope through its appropriate substring $\langle x_{m+1}, x_{m+2}, \ldots, x_n \rangle$. The Ag–Ab reaction was simulated as usual in this class of models based on binary sequences [26,27].

Lymphocytes must be able to bind only to antigens and not to the host organism itself (e.g., to the mouse organs) which is achieved by means of a negative selection algorithm [5,6,8]. In the present model, the algorithm first generates random binary strings $\langle x_0, x_1, \ldots, x_{15} \rangle$ modeling Ab paratopes in B-cells. Paratope strings have equal lengths (16 bits) as the third substring in mouse organs modelling histocompatibility antigens. Next, we calculated the Hamming distance, D_H^O, between the binary string x_j in the paratopes of B-cells and the strings x_j^O that represent the histocompatibility antigen of each organ:

$$D_H^O = \sum_O \left| x_j - x_j^O \right| \tag{5}$$

The goal was to generate strings (Ab) that did not recognize the strings representing the mouse organs. In the model, and in this paper, we applied the key-lock principle and, therefore, there was recognition that when in one position of a binary string there is a 0 and there is a 1 in the homologous position of the other string or vice versa, i.e., 1 and 0. Therefore, the Hamming distance between each string and the string of each organ must be minimal, summing the Hamming distances into a total Hamming D_S distance:

$$D_S = \sum_O D_H^O \tag{6}$$

If the total Hamming distance was less than a given self-recognition threshold θ, then we selected the resulting string:

$$D_S \leq \theta \tag{7}$$

Since in the mouse model there are 12 organs defined (O = 1,…,12) and assuming as a criterion a maximum Hamming distance per organ of two units, we set a recognition threshold value equal to 24. The following boxes (Appendix A, Boxes A1 and A2) show the pseudocode of the described algorithm. Applying N times the negative selection algorithm, we obtained a population of N lymphocytes composed of T-cells. Following, the lymphocyte population is stored (Appendix A, Box A2).

2.3.2. Humoral Immune Response: Clonal Selection Algorithm

According to the theory of clonal selection introduced by Burnet in 1959 [8], only those lymphocytes capable of recognizing an antigen (e.g., virus) are selected and multiplied to the detriment of the remaining lymphocytes. In other words, there is a clonal expansion of lymphocytes, in particular B-cells, increasing numerically those B-cells with greater affinity for the antigens. In

addition, such cells will gradually increase their affinity for antigens. Finally, once the clonal expansion is finished, some B-cells will become memory cells allowing the acquisition of immunity against future intrusions of similar antigens. In addition, other B-cells will secrete antibodies that neutralize (i.e., Ag–Ab) the antigen, i.e., virus.

The clonal selection algorithm [28–31] comprises three main steps: Ag–Ab recognition, clonal selection, and hypermutation (Box A6). Firstly, the affinity between the binary string x_{ij} modeling the Ab paratope carried by i B-cell (j is the bit position in the string) and Ag epitope x_j^V (i.e., the virus) is calculated by the Hamming distance. Based on this distance, we calculated the fitness of each B-cell being k a proportionality constant:

$$fitness_i^{B-Cell} = k \cdot \sum_j \left| x_{ij} - x_j^V \right| \quad (8)$$

In second place, the clonal selection was simulated by applying the algorithm of the wheel parents selection described above (Section 2.2.). Third, and last, hypermutation was simulated. This mechanism occurs during clonal expansion by modifying through mutation the genes that encode the antibodies present in B-cells. The mutation is greater when the affinity between the Ag and Ab is lower. In other words, the higher the fitness value of a B-cell, the lower the mutation probability in the Ab carried by the B-cell. The Ab mutation rate η is calculated by the following function:

$$\eta = \exp(-\frac{fitness_i^{B-Cell}}{\delta}) \quad (9)$$

where δ is a parameter for the exponential decay setting. If a B-cell is chosen to mutate, whether a bit experiment or not, mutation in its Ab is decided according to a given hypermutation rate. The clonal selection algorithm, together with the negative selection algorithm, allow us to consider the lymphocytes (i.e., B-cells and T-cells) as natural classifiers. The pseudocode of the described algorithms is depicted in Boxes A3–A7 (Appendix A).

2.3.3. Theory of Danger: The Dendritic Cell Algorithm

In 1994, Matzinger [8] proposed the theory of danger according to which the immune system is not triggered by the detection of, for example, a virus as it is the case with the humoral immune response paradigm. Once a virus enters into the mouse and its cells are damaged, a series of signals called danger signals are produced in response to the virus. From a biological point of view, the central idea is that only harmful intruders produce cell necrosis, i.e., the pathological death of the cell due to the fact of serious injury. We let the virus enter into the mouse and, according to the damages suffered by the mouse cells, it is decided whether or not to trigger the humoral immune response (algorithm of clonal selection, Section 2.3.2.).

The theory of danger was the inspiration for the dendritic cell algorithm (DCA) introduced by References [32,33]. A particular type of cell known as a dendritic cell (DC) is capable of receiving signals from both the inside and outside of the mouse cells. Dendritic cells are in three possible states: immature DC, semi-mature DC, or mature DC. Under normal conditions, the DCs are in an immature state monitoring the mouse cells. Therefore, a DC acts as a processor that receives input signals, processes them and sends output signals. Input signals can be of three types: pathogen-associated molecular patterns (PAMP) (PS), danger (DS), and safe (SS) signals. The first input signal reports the presence of harmful intruders (i.e., virus), and the second input signal notifies of damage in the mouse cells infected by the virus. Finally, the third input signal announces that the mouse cell is healthy (i.e., the possible alterations present in the cell are regarded as being within the normal limits of the cell). According to References [32,33], the inflammation mechanism (IC) has an amplification effect on the received signals. Once the input signals are processed and whether there is inflammation or not, the output signals are obtained. Output signals can be of three classes: co-stimulatory signals (O_{csm}), semi-mature signals (O_{semi}), and mature signals (O_{mat}). The first output signal is related with the migration of the DC to the lymphatic node (i.e., an organ of the mouse immune system). Once a DC reaches the lymphatic node, the clonal selection algorithm will be activated (see Section 2.3.2.).

The second kind of output signal reports that the mouse cellular environment is "safe". Finally, the third type of output indicates the opposite, i.e., that the mouse cellular environment is in "danger".

The processing of information in a DC takes place according to the following algorithm [34–36]. As long as the processing of the input signals leads to an O_{csm} output below a given threshold, the processing of the input signal continues. Once the threshold is reached, then the levels of the output signals O_{semi} and O_{mat} are compared. If the O_{semi} is greater than O_{mat}, then DC becomes semi-mature, i.e., the mouse cell is in a "safe" or "no danger" state. Otherwise, if O_{semi} is lower than O_{mat}, then DC reaches the mature DC state, emigrating from the damaged cell to the lymph node. Once the DC cell arrives at the lymphatic node, it triggers the humoral immune response, i.e., the mouse cells are in a "danger" state. The activation of the immune system takes place via cytokines (i.e., small proteins that have effects on other cells) that activate the T-cells, in the present model, by turning on the clonal selection algorithm. In the standard dendritic algorithm [32,33], the described model is expressed in terms of the following expressions:

$$\begin{aligned} O_{csm}(t) &= O_{csm}(t-1) + \left[PS.w_{11} + DS.w_{12} + SS.w_{13} \right].(1+IC) \\ O_{semi}(t) &= O_{semi}(t-1) + \left[PS.w_{21} + DS.w_{22} + SS.w_{23} \right].(1+IC) \\ O_{mat}(t) &= O_{mat}(t-1) + \left[PS.w_{31} + DS.w_{32} + SS.w_{33} \right].(1+IC) \end{aligned} \tag{10}$$

and setting the values of the weights w_{11}, w_{12},…, w_{33} according to the problem on which the algorithm is applied. The pseudocode of the described algorithm is shown in Box A8 (Appendix A).

2.4. WHD Model Architecture

The simulation model (Figure 3) relies on the hybridization of a simple genetic algorithm (virus evolution) with an artificial immune system (mouse defense system). Note how the artificial immune system model is a layered model as shown in Figure 3. A simulation experiment starts with virus detection once the virus has entered within the mouse (Figure 4). If the selected paradigm is the humoral immune response, then the virus detection algorithm is simulated with the clonal selection algorithm shown in Box A9 (Appendix A). In accordance with Box A7, the clonal selection algorithm selects the optimal B-cell with maximum fitness and, therefore, a greater affinity with the virus. Afterwards, the Ag (virus)–Ab (B-cell) affinity is calculated with the Hamming distance. Based on the distance obtained, we evaluate whether it is equal or greater than a given affinity threshold value—10 in the present simulation experiments. In addition, if a predefined stop generation is reached (100 in the current experiments) then the algorithm simulates the Ag (virus)–Ab (B-cell) reaction.

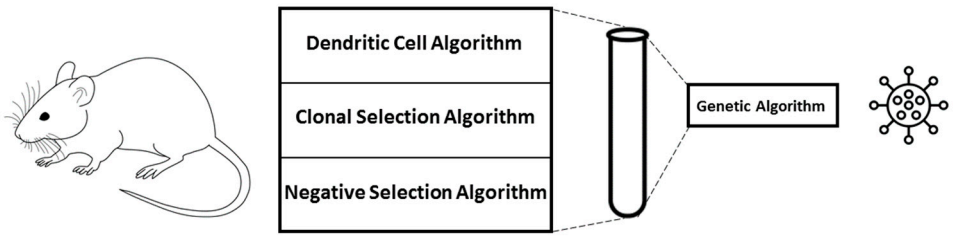

Figure 3. Hybridization of a genetic algorithm (virus evolution) with an artificial immune system (mouse defense system).

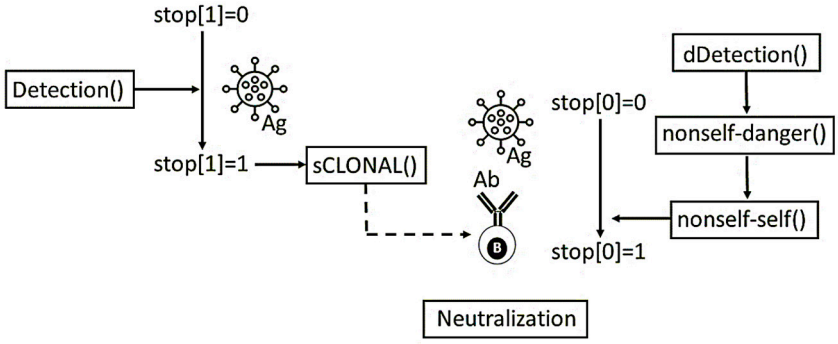

Figure 4. Stop signals (stop[0] or infection and stop[1] or detection) play the role of flags which state change as a result of the Detection() and dDetection() in nonself-self() and nonself-danger() functions. Function name nonself-self() refers to humoral immune response. In the field of AIS algorithms, *nonself* refers to the virus whereas *self* stands to the mouse. Likewise, nonself-danger() is a function name which refers to the theory of danger. State change in stop[1] (detection) signal triggers the clonal selection algorithm or sCLONAL() function, resulting in the virus neutralization reaction. The virus stands for antigen (Ag) and B-cells carry antibodies (Ab).

However, in the case of the theory of danger paradigm, virus detection (Figure 4) is held via a danger detection algorithm (Box A10, Appendix A). In this case, the dendritic cell algorithm (Box A8) receives the input signals (PS, DS, and SS) and process the input by means of Equation (10). Signals are processed until the O_{csm} value reaches a threshold value. When this threshold value is reached, then the O_{semi} and O_{mat} values are compared, and it is decided whether the DCs migrate from the mouse cell under surveillance to the lymphatic node. Box A10 shows an elementary model of the PS, DS, and SS signals. In the viral infected mouse cells signals concentrations are updated for each generation. If the levels of PS or DS exceed predefined thresholds, then inflammation mechanism is triggered, setting the inflammation coefficient (IC) to a particular value.

Figures 5 and 6 show the coupling between the GA and AIS algorithms in the humoral immune response and theory of danger paradigms, respectively. Note how, in both cases, the hybridization among the two algorithms is in the Ag (virus)–Ab (B-cell) neutralization step. Neutralization reaction simulation was conducted as follows. We generated a random number u, $u \in [0,1]$, that was compared with a neutralization rate or Ag (virus)–Ab (B-cell) reaction probability for which the value N_r was previously set:

$$\begin{cases} u \le N_r, F_i = 0 \\ u > N_r, F_i \ne 0 \text{ (4)} \end{cases} \quad (11)$$

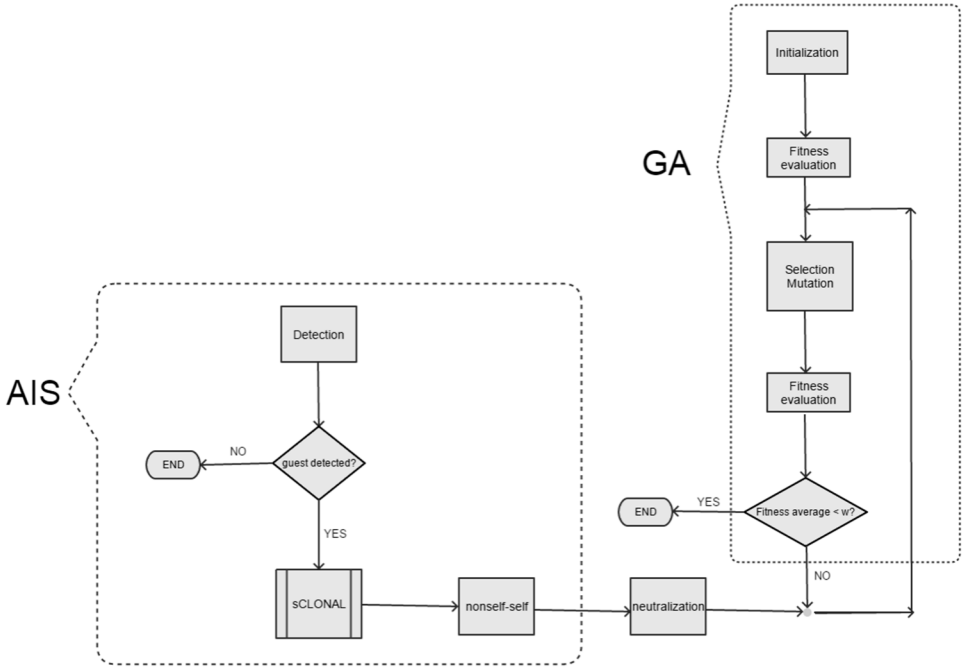

Figure 5. Hybridization between AIS and GA in the humoral detection algorithm (nonself-self).

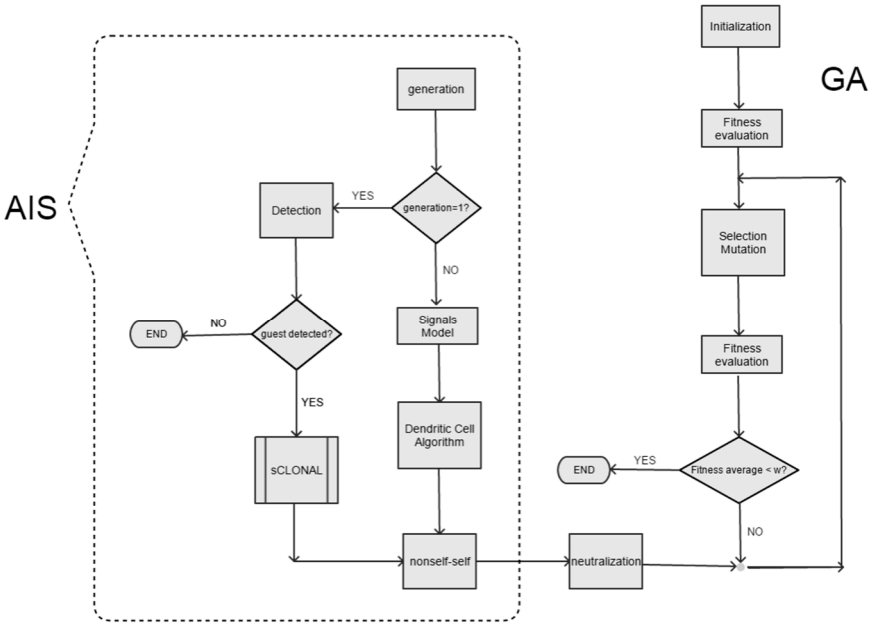

Figure 6. Danger detection (nonself-danger) and its implementation by hybridizing AIS with a GA.

If the B-cell successfully neutralizes the virus ($u \leq N_r$), then the fitness F_i of the virus would be zero. Otherwise, if the virus is not effectively neutralized by B-cells ($u > N_r$), then the viral fitness is calculated according to the Expression (4). Finally, it is evaluated whether the average fitness of the viral population is equal or less than a minimum value ω (minimum fitness average). In such a case, we assume that the virus is successfully neutralized by the immune system. Thus, it is presumed that,

under an evolutionary point of view below ω, the virus is not viable. In such a case, the simulation experiment ends—the virus is removed from the mouse.

2.5. Simulation Experiments

First, we performed a series of control simulation experiments, i.e., we simulated the evolution of the virus via a genetic algorithm in the absence of an immune system that prevents its evolution. The mutation rate of the virus population was equal to 0.1 with the probability of a bit mutation equal to 0.3. The experiments were carried out with 80 virions studying their evolution during 2000 generations.

Secondly, we studied the evolution of the virus under the influence of the immune system. In the experiments with the immune system in place, the virus population size was also equal to 80 virions, and the mutation operator parameters were similar to those used in the control experiments. However, we only studied 200 generations after experimentally finding that it is an adequate number of generations. In this work, we conducted three different types of simulation experiments. First, we simulated the negative selection algorithm. Afterwards we simulated the two paradigms of virus detection: humoral detection and danger detection paradigms.

In the humoral detection simulation experiments, the clonal selection algorithm evolved a number of B-cells equal to the number of virus, i.e., 80, being the hypermutation rate equal to 0.8 and the parameter of exponential decay δ = 0.5 (Equation (9), for B-cell selection). Although the clonal selection algorithm evolved B-cells over 500 generations, in previous experiments we observed that the selection of the optimal B-cell was achieved in a smaller number of generations. Several simulation experiments were performed in order to study the effects of the neutralization rate N_r and the minimum value of the average fitness ω.

For experiments conducted under the theory of danger paradigm, clonal selection was simulated under same experimental conditions and parameters as described above in the humoral detection experiments. In Equation (10), the values of the weights were a set of values that are usual in the dendritic cell algorithm: $w_{11} = 2$, $w_{12} = 1$, $w_{13} = 2$, $w_{21} = w_{22} = 0$, $w_{23} = 3$ and $w_{31} = 2$, $w_{32} = 1$, $w_{33} = -3$. The simulated signals model (Box A10) was a simple model in which we updated only the PAMP (PS) signals, simulating only the infection by a microorganism. At each generation, PS was increased: $\Delta \mathrm{PS} = 0.001$ units. When the level of PS was above 0.5 units, the inflammatory mechanism was triggered, setting a value IC = 1.0.

3. Designing a Computer Algorithm for Detection of Emerging DOS Malware: A Simple Example of WHD Model Application

In order to illustrate how the WHD model described in this paper can be used in computer science, we conducted the following experiment. In the example, we assumed the existence of DOS (Disk Operating System) programs with the ability to evolve within a computer assuming that as a result of this evolution a program that was initially useful to the computer can be transformed into a computer virus, malware, etc. [37]. We refer to these programs as "DOS malware".

Suppose a hypothetical computer (Figure 7) in its most elementary configuration with von Neumann architecture and that we label with the string $\langle 0,0,0,0 \rangle$. Therefore, the computer comprises a memory unit and a processor that we labelled as $\langle 0,0,0,1 \rangle$ and $\langle 0,0,1,0 \rangle$, respectively. In order to simplify the model as much as possible, we did not include the control unit, the arithmetic logic unit (ALU), accumulators, etc. In the computer memory there was a DOS program consisting of two code segments, chosen from a set of 16 possible code segments. Table 1 shows 16 code segments labelled with a 4-bit string and their fitness value. The fitness scores or values were assigned taking into account that a fitness value is higher the greater the maliciousness of the code. Therefore, codes 1010, 1011, 1100, 1101 and 1110 have a low fitness value, since they do not threaten the computer integrity, performing useful or leisure tasks. In contrast, codes 0010 and 0011 will cause serious damage to the computer. In the experiment, we assumed that initially in the memory of the computer there was a DOS utility program, its code being the result of the merging of segments 1100

and 1110. Running the file on the computer displays the current date and time, followed by the browser opening and loading a search engine page. We generated an initial population of 80 similar 11001110 programs. At any given moment, the DOS programs evolved by natural selection in the memory $\langle 0,0,0,1 \rangle$ of the computer, transforming into DOS malware. We assumed for the simplicity of the experiment that once the DOS program transformed into a malicious code, the resulting computer virus was not a polymorphic code that was capable of changing to evade detection by the computer, for example, from an antivirus or any other detection system.

Table 1. DOS (Disk Operating System) program: code segments and segment scores.

0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
50	40	60	80	50	40	20	30	20	20	10	10	10	10	10	20

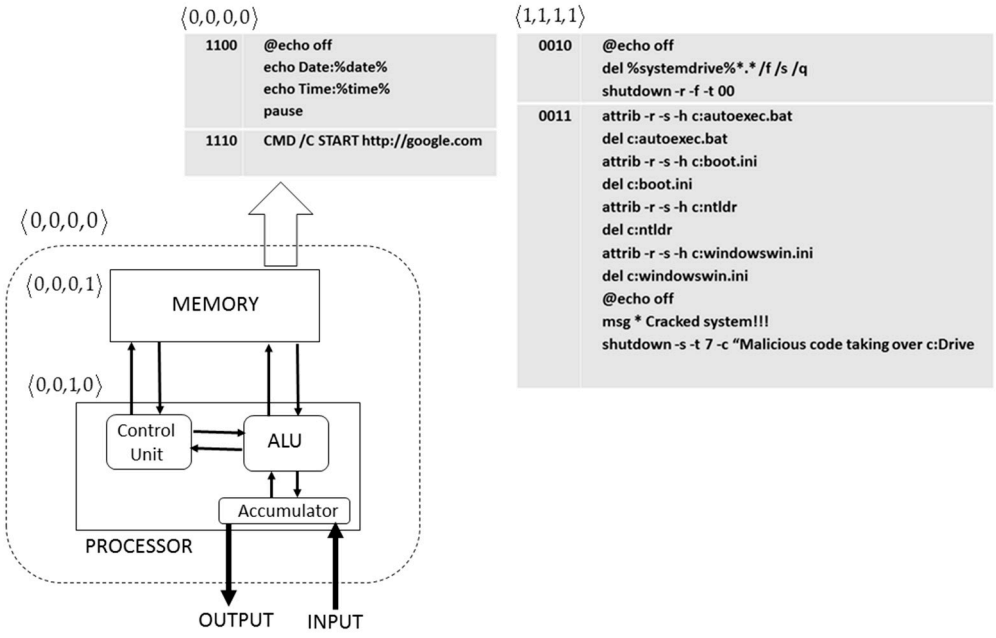

Figure 7. Model in which a computer is the host and the guest is a malicious DOS program that evolved from an initially useful DOS program (for an explanation see text).

In the simulation experiments, the programs evolved according to a simple GA. Each program was represented by a vector or chromosome of 8 bits being coded by two segments of the code in Table 1. The algorithm used a mutation operator as a variability source, choosing the population and bit mutation probabilities equal to 0.08 and 0.04, respectively. The fitness F_i of an individual i of the population (i.e., a DOS program) was obtained according to the following expression:

$$F_i = css_1.w_1 + css_2.w_2$$

with css_1 and css_2 being the score received for each code segment (Table 1) and w_1 and w_2, respectively, the weight values (w_1, w_2 = 10 in the experiments). Once the DOS programs were evaluated, the selection of those that would pass to the next generation was conducted by the wheel parent selection method [1,3] (see Section 2.2.). Eventually the population of programs converged to a program consisting of 0010 and 0011 segments. From a theoretical point of view, we considered the software, in our case the DOS program, as another computer component. For this reason, we initially assigned a 0000 label to the program 11001110. However, once this program evolved into malicious software, its label changed from $\langle 0,0,0,0 \rangle$ to $\langle 1,1,1,1 \rangle$, denoting that it was a harmful code and not a useful component of the computer. Notice how, from an immunological point of view and according to the

nomenclature described in Figure 4, that this was an experiment in which a *self*-program was transformed into a *nonself* program. An example of this type of change is the transformation of healthy cells into cancerous cells [38].

In this work, we performed two kinds of experiments: one in which the DOS program evolved in the absence of an algorithm that opposed their evolution and another in which the algorithm that was against the program's evolution was present in the computer. In the second case, the evolutionary algorithm that evolved the DOS program was confronted with an artificial immune system algorithm that expressed the computer's identity (*self*). We named the program that implemented this second algorithm "tiny clonal antivirus" (TCA). First, TCA detected the presence of a malignant code (*nonself* program) labeled $\langle 1,1,1,1 \rangle$, applying the humoral immune response paradigm (Figure 4). The detection was carried out by measuring the Hamming distance between the intruder program labeled $\langle 1,1,1,1 \rangle$ and the computer labeled $\langle 0,0,0,0 \rangle$. Since the distance was a maximum of (H = 4), the stop signal stop[1] = 0 changed to the state stop[1] = 1 (positive detection). Secondly, the change of state of the stop signal triggered the clonal selection algorithm or sCLONAL(), neutralizing the DOS malicious programs. The parameters of the sCLONAL() function were similar to those used in Section 2.5. The function nonself-self() was tailored to the present problem. In consequence, the change of state of the signal stop[0] = 0 to stop[0] = 1 (positive infection) occurs when the paratope sequence in the B-cell has maximum affinity (maximum Hamming distance) with the epitope of a malicious program, i.e., 0010 or 0011.

4. Results

In general, we conclude that the simulation experiments reasonably captured the evolutionary dynamics of the guest (i.e., virus, DOS programs) within a host (i.e., mouse, computer) that fights against guest evolution. The hybridization of genetic algorithms with artificial immune system algorithms allows us to conceive evolutionary WHD models for simulating scenarios that, to date, has not been modelled with differential equations.

We therefore conclude that the proposed evolutionary WHD model adequately simulates the coexistence of two agents exhibiting both an evolutionary dynamic against each other.

4.1. Control Experiment

Figure 8 shows the evolution of the virus in the control experiment (i.e., in the absence of the immune system) such as, for example, of a virus within an immuno-depressed host. In this case, we obtained the classic performance graph of a genetic algorithm.

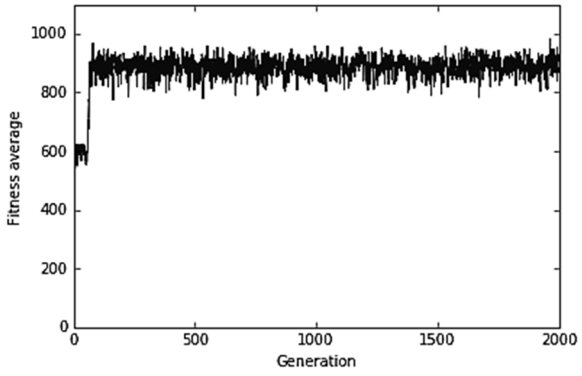

Figure 8. Control experiment: virus evolution when the immune system is lacking.

4.2. Thymus Model and Simulation: Negative Selection Algorithm

The simulation of the mouse thymus (i.e., the organ where the negative selection algorithm occurs) adequately yielded to mature lymphocytes, e.g., B-cells. Such cells displayed the feature that they do not bind to mouse organs. For example, in one of the simulation experiments, we obtained

twelve B-cells: 0101010101001000, 0101010101000001, 0101010101001011, 0101010101000111, 0101010101001001, 0101010101000000, 0101010101000010, 0101010101000101, 0101010101000011, 0101010101000111, 0101010101000110, and 0101010101001001.

Although this algorithm was not used in later experiments, it was part of the current study, as it may be of interest to immunologists, bioinformatics, etc.

4.3. Humoral Detection and Immune Response

Figures 9 and 10 shows representative results obtained in the humoral immune response simulation experiments. In one of the experiments (Figure 9) with a neutralization rate of 25% (N_r = 0.25) and minimum fitness value ω = 50, it was observed that the immune system was not successful and the virus was not removed from the host. Therefore, in this experiment, the virus evolved in a stepped way until reaching the optimal genome. However, in another experiment (Figure 10), when the neutralization rate was high up to 99% (N_r = 0.99) and the minimum fitness value was ω = 50, the virus was successfully cleared from the mouse. Figure 10 shows how the average fitness of the virus population declined to zero in generation 101. However, in Figure 9, the Ag–Ab neutralization reaction that takes in generation 100 was unsuccessful. Virus average fitness fell in generation 101, drawing a valley in the performance graph of the virus evolution. However, once the neutralization reaction was over, the virus population recovered and continued its evolution.

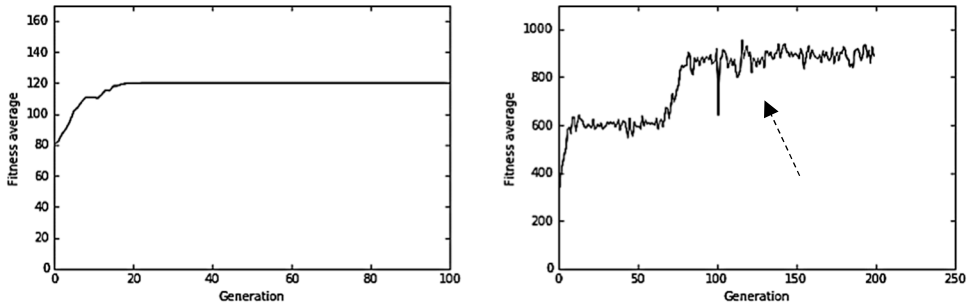

Figure 9. Humoral detection experiment with N_r = 0.25 and minimum fitness value ω = 50. The figure on the left shows the evolution of B-cells because of the clonal selection algorithm. The figure on the right depicts the evolution of the virus within the host. Note how the virus successfully evolved, escaping the effects of the immune system (after the generation indicated by the arrow).

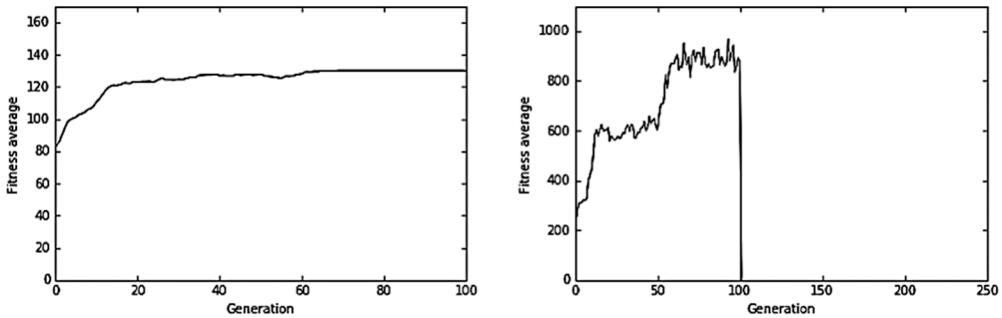

Figure 10. Humoral detection experiment with N_r = 0.99 and minimum fitness value ω = 50. The figure on the left shows the evolution of B-cells because of the clonal selection algorithm. The figure on the right depicts the evolution of the virus within the host until it was finally cleared from the host by the action of the immune system.

The minimum fitness value ω has an effect on the dynamics of the virus, which depends on the neutralization rate N_r. Studying different values of ω and N_r, we found that given some value of ω, a successful clearance of the virus occurred when the neutralization rate was equal or greater than 95%.

4.4. Danger Detection and Immune Response

Figure 11 shows an illustrative result obtained in the experiment based on the theory of danger. Although the viral dynamics seemed similar to that obtained for the humoral system (Figures 9 and 10), the underlying mechanism was more complicated, because it represents a higher-level response from the immune system. In this case, the virus was neutralized once the PAMP (PS) signal reached a given level in the host cells. In the experiment, the PS signal increased generation after generation to a certain extent. Signal increase will be the cause of the virus neutralization reaction, which takes place in generation 100 and virus clearance in the next generation, i.e., 101.

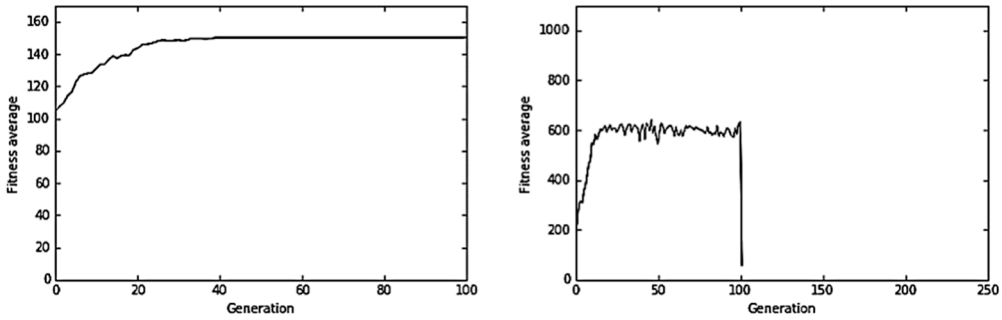

Figure 11. Danger detection experiment with $N_r = 0.99$ and minimum fitness value $\omega = 50$. The figure on the left shows the evolution of B-cells because of the clonal selection algorithm. The figure on the right depicts the evolution of the virus within the host until the PAMP signal reaches a given value. Once the dendritic cell algorithm processes the PAMP signal, the dendritic cells migrate from the tissue to the lymphatic node triggering the humoral immune response—the virus is removed from the host.

It is worth noting that when the virus is successfully eliminated in both paradigms (humoral detection and danger detection), the performance graph (Figures 9–11) resembles the numerical solution (Figure 12) of the number of virus V per generation obtained from the ordinary differential equations [15].

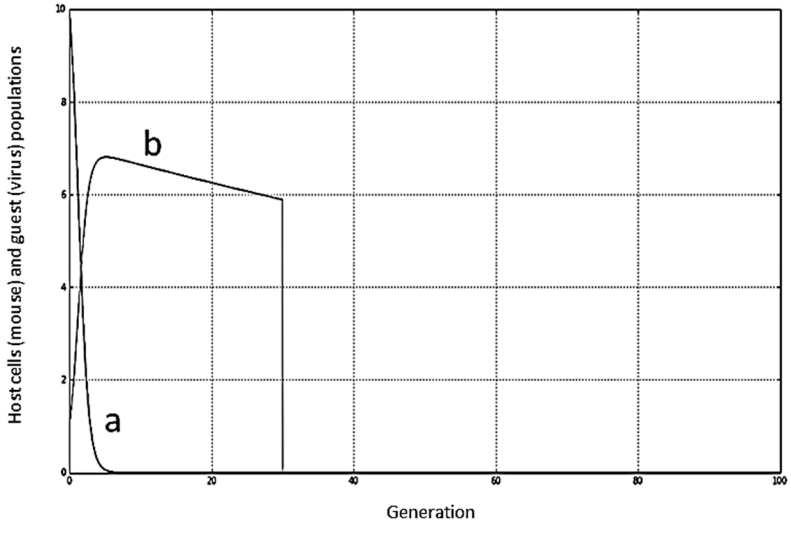

Figure 12. Numerical solution of ordinary differential equations [15] with initial conditions $T(0) = 10$ and $V(0) = 1$ and parameters $\beta = 0.2$, $r = 0.6$, and $\gamma = 0.006$. The curve that decays exponentially (**a**) is the number of host cells susceptible to being infected, and the curve of rectangular appearance (**b**) is the number of viruses by generation. It has been simulated that, for a generation number equal or greater than 30, the viruses are successfully neutralized with the population size falling to a value $V = 0$.

4.5. DOS Malware Detection Simulation

Figure 13 shows the evolution of DOS programs into malicious DOS programs when the computer (Figure 7) is not protected by the TCA algorithm. However, when the TCA program is present, the DOS malware is detected. Once detected, it is successfully removed from the computer memory (Figure 14).

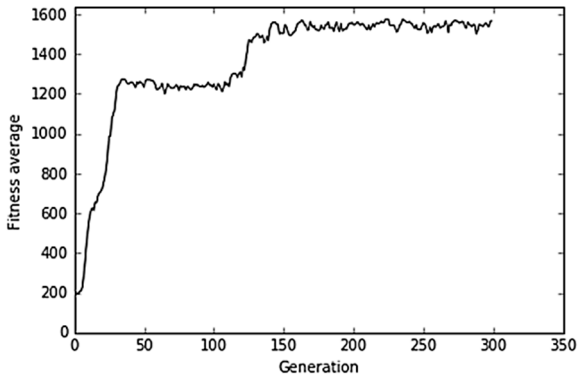

Figure 13. Evolution from DOS programs to DOS malware when the TCA program is lacking**.**

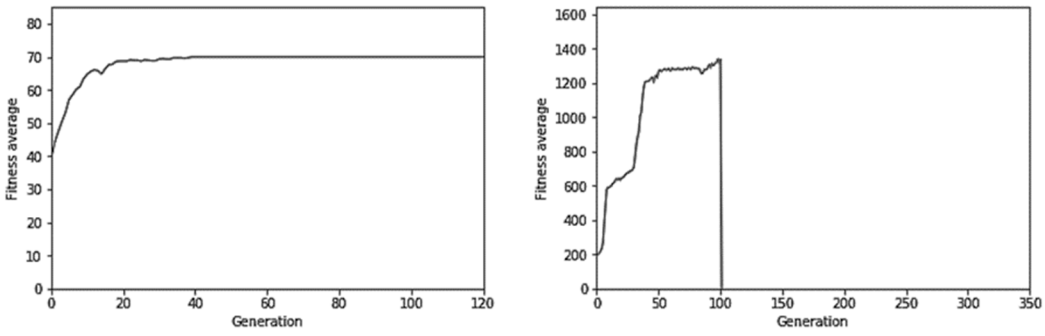

Figure 14. DOS malware detection experiment. The figure on the left shows the evolution of B-cells as a result of the clonal selection algorithm. The figure on the right depicts the evolution of the DOS programs within the host. Once the DOS programs are transformed into DOS malware and detected by the TCA program, they are cleared from the host by the action of the immune system.

5. Discussion

The main goal of this paper was to hybridize a genetic algorithm with artificial immune system algorithms in order to obtain an environment in which two agents coexist and evolve against each other. Indeed, the coexistence of two agents evolving against each other is a theoretical situation in which it is possible to conduct a very large number of simulation experiments. Our goal was to design a general framework for the study of an evolutionary WHD model, and for this reason, we have not conducted all possible simulation experiments. This work therefore represents a first step in achieving this goal. In our opinion, our model shows how using bioinspired algorithms (i.e.,

evolutionary computing algorithms) we can simulate detailed mechanisms which it is not possible to simulate with ordinary differential equations. Among many other reasons, the parameters of a model stand for quantitative abstractions of a given phenomenon or mechanism, e.g., the differential equations model the viral mortality rate γ. In consequence, in ordinary differential equations models, many relevant details are not included, e.g., all immunological processes—algorithms responsible for viral clearance. Another novel feature of our approach is host modeling. The approach developed in this paper allows us to simulate both the effects of the immune system and the evolutionary plasticity of the immune system as the virus evolves. In addition, with our approach, it is possible to simulate different guest detection mechanisms, immune system action levels, neutralization reaction, etc.

Many other simulation experiments could be conducted, for example, the simulation of the evolutionary mechanisms developed by the virus to attack the immune system of the mouse, destroying the host defenses. In addition, a replication mechanism inspired by RNApolymerase, an important enzymatic machinery responsible for self-replication of RNA viruses, could have been included in the present model. Of course, all these ideas will be of interest to virologists, immunologists, and bioinformaticians interested in the study via simulation of the host–parasite interaction. However, beyond a possible clinical application of the proposed model, our model could also be of interest in other areas such as computer science, for instance, in the design of anti-virus software such as intrusion detection algorithms in a corporation's computer systems, especially in those cases where the invader exhibits evolutionary changes. In this paper, we simulated an experiment in which a DOS program that is initially useful to a computer evolves into DOS malware. Furthermore, the example illustrates how to encode the problem under study within the methodology we have developed. In general, the coding of the problem consists of tailoring the genetic algorithm that simulates the evolution of the guest and customizing the AIS routines [39], written in Python, to the problem.

As the complexity of communication networks increases, computers and communications systems will require intelligent systems capable of detecting malware with evolutionary traits. Therefore, a possible strategy will be to foresee these scenarios by simulating such environments by the hybridization between genetic algorithms and artificial immune systems. That is, and because of this hybridization, the integration of AIS in the host explains its detrimental effects on the evolution and neutralization of the guest. In this paper, the hybridization technique (Section 2.4.) between the GA and the AIS is the most natural, just as it happens in biology, considering the bio-inspired flavor of the model. Note how the guest detection, the immunological paradigm (i.e., humoral immune response and theory of danger), and neutralization reaction are the most relevant steps for linking both algorithms. In fact, but with a different purpose, the implementation of the hybrid AIS–GA is not a novel approach. Nevertheless, there are previous publications in which their hybridization is addressed, but the main goal was to increase the individual efficiency of both algorithms by collaborating in a common optimization problem [40–42]. For example, Reference [43] found that, depending on the benchmark function, sometimes the clonal selection algorithm provides a better performance than genetic algorithms, but in other cases the genetic algorithm is the one with the best performance. In our model, the novel feature is that these two kinds of algorithms do not collaborate, but they compete with each other. Thus, the evolution and increase of fitness in the virus (guest) is at the expense of decreasing the fitness of the host—the mouse. In a similar way, and by adopting an evolutionary perspective, we could think that it happens with a computer virus (guest) and an infected computer (host). However, our model does not include the calculation of the host fitness or a mathematical model calculating host fitness variation in function of the guest fitness. As future work, we propose the designing of an antivirus algorithm based on artificial immune system [44,45] but including the evaluation of the computer virus danger (increased guest fitness) depending on the damage that it would cause in the computer (decreased host fitness).

In order to make this paper readable, and as we have already mentioned before, we have not included a detailed study of the parameter values' effect on the simulation results. We have not studied all the possible values of the weights in Expression (10) of the model that implements the dendritic cell algorithm. In addition, it would be interesting to study the different models of PS, DS,

and SS signals and include a mathematical model with enough detail of the inflammatory mechanism. Since there is an equivalence between biological concepts and their counterparts in computer science, we plan as future work to address the above issues, as they are fundamental in the design of intrusion detection systems. Nevertheless, in both the GA and the AIS algorithms, the most suitable values of many other parameters are known from preliminary simulation experiments. In fact, some parameters are common to both kinds of algorithms: GAs and AIS share some procedures, for example, mutation and hypermutation. In fact, these two kinds of algorithms belong to related families and are used in related problems. Moreover, GAs and AIS speak the same computing language by processing with similar mechanisms strings of symbols.

Obviously, the model of virus–mouse coexistence introduced in this paper is a metaphor that could help design bioinspired solutions in other scientific and technological fields. We are referring to those scenarios in which two agents coexist and evolve simultaneously against each other. Not only in biology or computer science does this situation arise, but economics and politics are also suitable scenarios for the application of our model.

Author Contributions: A.G.-M. has collaborated in the discussion, revision and carried out simulation experiments that were used in his Master's Degree in Industrial and Environmental Biotechnology 2018–2019, Complutense University of Madrid. R.L.-B. has devised the general problem, the model and written the Python routines of the artificial immune system and genetic algorithm. He has also supervised the work of the first author and written this paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: R.L.-B. also thanks to the Department of Genetics, Physiology and Microbiology and the Departmental Section of Biochemistry and Molecular Biology for their collaboration through their Master's students with the Biomathematics Unit of the Faculty of Biology, Complutense University of Madrid.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Simulation experiments were conducted using an artificial immune system library written in Python: mais.py (Mouse Artificial Immune System). The AIS library and genetic algorithm can be downloaded from Reference [39]. Below we show in pseudocode the main routines of the artificial immune system.

Box A1. Negative selection algorithm.

```
Negative_selection(filename)
        for j=0 to paratope_length do
          u=generate_random(.)
              if u≤0.5
                then
                      T-cell[cells,j]=0
                  else
                      T-cell[cells,j]=1

        for i=0 to organs do
          for j=0 to paratope_length do
            m=T-cell[cells,j]
             n[i]=mhc(host[i,j])
             Ds=affinity(m,n[i])
             if Ds≤detection_threshold
              then
                  break
        create_file(filename)
```

Box A2. Thymus simulation algorithm.

```
Set_Tcell(filename, N)
  i=0
  while i≤N do
    Negative_selection(filename)
    i=i+1
    if i == N
        then
          break
```

Box A3. Clonal selection algorithm: Ag-Ab recognition step.

```
AgAb_recognition()
    m=B-cell[i,j]
    n=epitope(virus[j])
    a=affinity(m,n[i])
    fitness_B-cell=f(m)
```

Box A4. Clonal selection algorithm: Clonal selection step.

```
Clonal_selection()
    u=generate_random(.)
    sum=0
    for i=1 to popSize do
        sum=sum+P(choice=i)
        if u<sum
          then
            return i
```

Box A5. Clonal selection algorithm: Hypermutation step.

```
mutation(mutation_rate, delta)
  for i=1 to popSize do
    u=generate_random(.)
      if u≤exp(-fitness_B-cell/delta)
        then
          for j=0 to paratope_length do
              v=generate_random(.)
              if ≤mutation_rate
                then
                  0 → 1
                  1 → 0
  Replace last B-cell population withn new individuals
```

Box A6. Clonal selection algorithm.

```
sCLONAL()
    seed()
    generation=0, generation_max
    Init_BcellsPopulation
    Virus(.)
    AgAb_recognition
    while generation≤generation_max do
        Clonal_selection
        generation=generation+1
        AgAb_recognition
        mutation(0.5,5)
```

Box A7. Nonself-self detection algorithm.

```
nonself_self(rule, threshold, stop_generation, generation)
    stop[0]=0
    Ab = selects_best(B-cell[i,j])
    Ag = epitope(virus[j])
    a = affinity(Ag, Ag)
    if rule=0
     then
        if a=threshold and generation=stop_generation
         then
           stop[0]=1
    if rule=1
     then
        if a=threshold or generation=stop_generation
         then
           stop[0]=1
```

Box A8. Nonself-danger detection algorithm.

```
nonself_danger(tissue, PS, DS, SS, IC, threshold)
    organ=f(tissue)
    Ocsm[i]=0, Osemi[i]=0, Omat[i]=0, DCit=100
    for i=0 to DCit do
        if Ocsm[i]<threshold
          then
            Ocsm[i]=Ocsm[i-1]+(PS*w11+DS*w12+SS*w13)*(1+IC)
            Osemi[i]=Osemi[i-1]+(PS*w21+DS*w22+SS*w23)*(1+IC)
            Omat[i]=Omat[i-1]+(PS*w31+DS*w32+SS*w33)*(1+IC)
      else
          if Osemi[i]>Omat[i]
           then
             Dcell[organ]=0
           else
             Dcell[organ]=1
```

Box A9. Humoral detection algorithm.

```
Detection():
    stop[1]=0
    virus (guest) enters host
    randomly selects host organ
    'compares virus epitope with fixed sequence
    (third list) of selected host organ'
    if   'the two sequences do not match'
        then
          'foreign agent detected'
          stop[1]=1
    if stop[1]=1
      then
        sCLONAL()
```

Box A10. Danger detection algorithm.

```
dDetection():
   if generation=1
     then
       Detection()
   else
     'PS, DS, SS signals model'
      PS[organ, generation]=PS[organ, generation -1]+ Δ PS[organ, generation -1]
      DS[organ, generation]=DS[organ, generation -1]+ Δ DS[organ, generation -1]
      SS[organ, generation]=SS[organ, generation -1]+ Δ SS[organ, generation -1]
      if PS[organ, generation]>ps_threshold or DS[organ, generation]> ps_threshold
       then
         IC[organ]
       if DCell[organ]=1
        then
          nonself_self(rule, threshold, stop_generation, generation)
```

References

1. Lahoz-Beltra, R. *Bioinformática: Simulación, Vida Artificial e Inteligencia Artificial*; Ediciones Díaz de Santos: Madrid, Spain, 2004, pp. 237–323. (In Spanish)
2. Salas Machado, R.; Castellanos Peñuela, A.L.; Lahoz-Beltra, R. Eye evolution simulation with a genetic algorithm based on the hypothesis of Nilsson and Pelger. *Int. J. Inf. Theor. Appl.* **2017**, *24*, 221–228.
3. Lahoz-Beltra, R. Evolving hardware as model of enzyme evolution. *BioSystems* **2001**, *61*, 15–25.
4. Perales Graván, C.; Lahoz-Beltra, R. Evolving morphogenetic fields in the zebra skin pattern based on Turing's morphogen hypothesis. *Int. J. Appl. Math. Comput. Sci.* **2004**, *14*, 351–361.
5. Aickelin, U.; Dasgupta, D.; Gu, F. Artificial immune systems. In *Search Methodologies. Introductory Tutorials in Optimization and Decision Support Techniques*, 2nd ed.; Burke, E.K., Kendall, G., Eds.; Springer: New York, NY, USA, 2014; pp. 187–212.
6. AISWeb. The Online Home of Artificial Immune Systems. Available online: www.artificial-immune-systems.org (accessed on 15.05.2019).
7. Castro, L.N.; Timmis, J.I. Artificial immune systems as a novel soft computing paradigm. *Soft Comput.* **2003**, *7*, 526–544.
8. Dasgupta, D.; Niño, L.F.; *Immunological Computation: Theory and Applications*; CRC Press, Auerbach Publications and Taylor & Francis Group: Boca Raton, FL, USA, 2009.
9. Timmis, J. Artificial immune systems—Today and tomorrow. *Nat. Comput.* **2007**, *6*, 1–18.
10. Wang, W.; Gao, S.; Li, F.; Tang, Z. A complex artificial immune system and its immunity. *Int. J. Comput. Sci. Netw. Secur.* **2008**, *12*, 287–295.
11. Fernandes, D.A.B.; Freire, M.M.; Fazendeiro, P.A.P.; Inácio, P.R.M. Applications of artificial immune systems to computer security: A survey. *J. Inf. Secur. Appl.* **2017**, *35*, 138–159.
12. Luo, X.; Wei, W. A new immune genetic algorithm and its application in redundant manipulator path planning. *J. Robot. Syst.* **2004**, *21*, 141–151.
13. Ciupe, S.M.; Heffernan, J.M. In-host modeling. *Infect. Dis. Model.* **2017**, *2*, 188–202.
14. Bocharov, G.A.; Telatnikov, I.S.; Chereshnev, V.A.; Martinez, J.; Meyerhans, A. Mathematical modeling of the within-host HIV quasispecies dynamics in response to antiviral treatment. *Russ. J. Numer. Anal. Math. Model.* **2015**, *30*, 157–170.
15. Hadjichrysanthou, C.; Cauët, E.; Lawarence, E.; Vegvari, C.; De Wolf, F.; Anderson, R.M. Understanding the within-host dynamics of influenza A virus: from theory to clinical implications. *J. R. Soc. Interface* **2016**, *13*, 20160289, doi:10.1098/rsif.2016.0289.
16. Cerf, R. The quasispecies regime for the simple genetic algorithm with ranking selection. *Trans. Amer. Math. Soc.* **2017**, *369*, 6017–6071.
17. Lauring A.S.; Andino, R. Quasispecies theory and the behavior of RNA viruses. *PLoS Pathog,* **2010**, *6*, e1001005, doi:10.1371/journal.ppat.1001005.
18. Heidtke, K.; Schulze-Kremer, S. Design and implementation of a qualitative simulation model of λ phage infection. *Bioinformatics* **1998**, *14*, 81–91.

19. Bocharov, G.; Ford, N.J.; Edwards, J.; Breinig, T.; Wain-Hobson, S.; Meyerhans, A. A genetic-algorithm approach to simulating human immunodeficiency virus evolution reveals the strong impact of multiply infected cells and recombination. *J. Gener. Virol.* **2005**, *86*, 3109–3118.
20. Flipse, J.; Wilschut, J.; Smit, J.M. Molecular mechanisms involved in antibody-dependent enhancement of dengue virus infection in humans. *Traffic* **2013**, *14*, 25–35.
21. Yeom, J-S.; Kostova-Vassilevska, T.; Barnes, P.D., Jr.; Tomas Oppelstrup, D.R.J. Exploratory modeling and simulation of the evolutionary dynamics of single-stranded RNA virus populations. In Proceedings of the 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lake Buena Vista, FL, USA, 29 May–2 June 2017; pp. 263–272.
22. Woo, H.Y.; Reifman, J. Quantitative modeling of virus evolutionary dynamics and adaptation in serial passages using empirically inferred fitness landscapes. *J. Virol.* **2014**, *88*, 1039–1050.
23. Jian, C.; Li, F. An improved virus evolutionary genetic algorithm for workflow mining. *J. Theor. Appl. Inf. Technol.* **2013**, *47*, 406–411.
24. Zhang, Y.; Li, T.; Qin, R. Computer virus evolution model inspired by biological DNA. In *International Conference on Intelligent Computing*; Huang D.S., Wunsch, D.C., Levine, D.S., Jo, K.H., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 943–950.
25. Van de Sandt, C.E.; Kreijtz, J.H.C.M.; Rimmelzwaan, G.F. Evasion of influenza A viruses from innate and adaptive immune responses. *Viruses* **2012**, *4*, 1438–1476.
26. Forrest, S.; Javornik, B.; Smith, R.E; Perelson, A.S. Using genetic algorithms to explore pattern recognition in the immune system. *Evol. Comput.* **1993**, *1*, 191–211.
27. Forrest, S.; Perelson, A.S.; Allen, L.; Cherukuri, R. Self-nonself discrimination in a computer. In Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, 16–18 May 1994; pp. 202–212.
28. Cutello, V.; Narzisi G.; Nicosia G.; Pavone M. Clonal selection algorithms: A comparative case study using effective mutation potentials. In *Artificial Immune Systems, ICARIS 2005 Lecture Notes in Computer Science*; Jacob C., Pilat M.L., Bentley P.J., Timmis J.I. Eds.; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3627, pp. 13–28.
29. de Castro, L.N.; Von Zuben, F.J. The clonal selection algorithm with engineering applications. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '00), Workshop on Artificial Immune Systems and Their Applications, Las Vegas, NV, USA, 8–12 July 2000; pp. 36–37.
30. de Castro, L.N.; Von Zuben, F.J. Learning and optimization using the clonal selection principle. *IEEE Trans. Evol. Comput.* **2002**, *6*, 239–251.
31. Brownlee, J. *Clonal Selection Algorithms*; CIS Technical Report 070209A, Complex Intelligent Systems Laboratory; Swinburne University of Technology: Melbourne, Australia, 2007.
32. Greensmith J.; Aickelin U. The deterministic dendritic cell algorithm. In *Artificial Immune Systems, ICARIS 2008*; Lecture Notes in Computer Science; Bentley, P.J., Lee, D., Jung, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; Volume 5132, pp. 291–302.
33. Greensmith, J.; Aickelin, U.; Cayzer, S. Detecting danger: The dendritic cell algorithm. In *Robust Intelligent Systems*; Springer: New York, NY, USA, 2008; pp. 89–112, doi:10.1007/978-1-84800-261-6_5.
34. Chelly, Z.; Elouedi, Z. A survey of the dendritic cell algorithm. *Knowl. Inf. Syst.* **2016**, *48*, 505–535.
35. Ding, L.; Yu, F.; Yang, Z. Survey of DCA for abnormal detection. *J. Softw.* **2013**, *8*, 2087–2094.
36. Zheng, X.; Fang. F. Principle and application of dendritic cell algorithm for intrusion detection. In *Proceedings of the 2011 3rd International Conference on Signal Processing Systems, ICSPS 2011*; IACSIT Press: Singapore, 2012; Volume 48, pp. 85–91.
37. Koret, J.; Bachaalany, E.; *The Antivirus Hacker's Handbook*; John Wiley & Sons: Indianapolis, IN, USA, 2015.
38. Måløy, M.; Måløy, F.; Lahoz-Beltra, R.; Nuño, J.C.; Bru, A. An extended Moran process that captures the struggle for fitness. *Math. Biosci.* **2019**, *308*, 81–104.
39. Lahoz-Beltra, R. mais.py (Mouse Artificial Immune System): A Python library for the hybridization of genetic algorithms with an artificial immune system. *Figshare Softw.* **2019**, doi:10.6084/m9.figshare.9758663.v5.
40. Coello, C.A.; Cruz Cortés, N. Hybridizing a genetic algorithm with an artificial immune system for global optimization. *Eng. Optim.* **2004**, *36*, 607–634.
41. Rajasekaran, S.; Lavanya, S. Hybridization of genetic algorithm with immune system for optimization problems in structural engineering. *Struct. Multidiscip. Optim.* **2007**, *34*, 415–429.

42. Mohammed Obaid, A.; Koh, S.P.; Chong, K.H.; Yap, D.F.W. Hybrid artificial immune system-genetic algorithm optimization based on mathematical test functions. In Proceedings of the 2010 IEEE Student Conference on Research and Development (SCOReD 2010), Putrajaya, Malaysia, 13–14 December 2010; pp. 256–261.
43. Ülker, E.D.; Ülker, S. Comparison study for clonal selection algorithm and genetic algorithm. *Int. J. Comput. Sci. Inf. Technol. (IJCSIT)* **2012**, *4*, 107–118.
44. Ali, H.A.; Hussain, D.J. Computer virus detection based on artificial immunity concept. *Int. J. Emerg. Trends Technol. Comput. Sci. (IJETTCS)* **2014**, *3*, 68–74.
45. Lee, H.; Kim, W.; Hong, M. Artificial immune system against viral attack. In *International Conference on Computational Science (ICCS 2004)*; Lecture Notes in Computer Science; Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3037, pp. 499–506.