Computational Power of Killers and Helpers in the Immune System

José Daniel Dias Pacheco

DI–FCUL TR–04–10

July 2004

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

COMPUTATIONAL POWER OF KILLERS AND HELPERS IN THE IMMUNE SYSTEM

José Daniel Dias Pacheco

Dissertação submetida para obtenção do grau de
MESTRE EM INFORMÁTICA

Orientador:

José Félix Gomes da Costa

Co-Orientador:

Helder Manuel Ferreira Coelho

Júri:

Jorge Orestes Cerdeira

Carlos Eduardo Ramos dos Santos Lourenço

Abril de 2004

COMPUTATIONAL POWER OF KILLERS AND HELPERS IN THE IMMUNE SYSTEM

José Daniel Dias Pacheco

Dissertação submetida para obtenção do grau de
MESTRE EM INFORMÁTICA

pela

Faculdade de Ciências da Universidade de Lisboa

Departamento de Informática

Orientador:

José Félix Gomes da Costa

Co-Orientador:

Helder Manuel Ferreira Coelho

Júri:

Jorge Orestes Cerdeira

Carlos Eduardo Ramos dos Santos Lourenço

Abril de 2004

Resumo

O sistema imunológico natural tem vindo a ser alvo de um grande interesse em termos de investigação devido às suas poderosas capacidades de processamento de informação. Utiliza características tais como aprendizagem, memória e recuperação associativa para resolver tarefas de reconhecimento e classificação.

O modelo apresentado neste trabalho pertence à classe de modelos introduzidos por Farmer *et al* e é inspirado na hipótese da teoria de selecção clonal da rede idiotípica introduzida por Niels Jerne. O objectivo principal é o de apresentar um Algoritmo Imunológico modificado que possa ser usado para resolver problemas de forma semelhante aos Algoritmos Evolucionários e às Redes Neuronais. Para além de apresentar o algoritmo própriamente dito são também discutidos os seus diversos parâmetros, a forma de apresentar problemas e como extrair resultados. O modelo é então descrito como sendo um meta-algoritmo probabilístico. São ainda apresentados diversos problemas reais de forma a comparar o modelo com outros modelos de algoritmos para a resolução de problemas de inspiração biológica. Finalmente são discutidas diversas métricas de forma a comparar a eficiência e os resultados dos diversos modelos de inspiração biológica.

Palavras chave
sistemas imunitários artificiais, algoritmos probabilisticos

Abstract

The natural immune system is a subject of great research interest because of its powerful information processing capabilities. It uses characteristics such as learning, memory and associative retrieval to solve recognition and classification tasks.

The model presented in this work belongs to the class of models introduced by Farmer *et al* and is inspired by the hypothesis of clonal selection theory and idiotypic network introduced by Niels Jerne. The main objective is to present a modified Immunological Algorithm that can be used in order to solve problems much in the way that Evolutionary Algorithms or certain types of Artificial Neural Networks do. Besides presenting the algorithm itself we discuss his various parameters, the way to present problems to it and how to extract results from its outcome. The model is then described as being a meta-algorithm to the Probabilistic Algorithms set. Several real problems are presented in order to compare this model with other types of Biologically inspired solving problems models. Finally we discuss various metrics to compare the efficiency and the results of the various Biologically inspired models.

Keywords
artificial immune systems, probabilistic algorithms

Aos meus pais, Daniel e Belmira,
a quem nem sempre soube dizer o quanto gosto deles.

Agradecimentos

Quero agradecer a todos aqueles que, directa ou indirectamente, contríbuiram para a realização deste trabalho. Este foi realizado com o apoio financeiro do Sub-Programa Ciência e Tecnologia do 2º Quadro Comunitário de Apoio, através da bolsa PRAXIS XXI/BM/6875/95.

Em particular, gostaria de agradecer ao meu orientador, José Félix Gomes da Costa, não só pelo apoio e confiança demonstradas, mas também pelos conselhos dados ao longo da realização deste trabalho.

Ao Luís Caires, Margarida Mamede e restantes colegas, pelo incentivo.

Ao Prof. João Bento, pelo enorme apoio e confiança.

Ao meu amigo Miguel Afonso, nunca serás esquecido.

Aos meus amigos, em particular ao Miguel, Francisco, Nuno, Tiago, Paleta e David, por estarem lá sempre que precisei deles.

Às minhas *crianças* da Faculdade de Ciências, pelo apoio e amizade.

Ao David Tibet, pela infindável fonte de inspiração.

Aos meus pais e irmão, por tudo.

Acknowledgments

I would like to thank all those who, directly or indirectly, contributed to the accomplishment of this work. This work was financially supported by "Sub-Programa Ciência e Tecnologia do 2º Quadro Comunitário de Apoio", with the grant PRAXIS XXI/BM/6875/95.

In particular, I would like to thank my supervisor, José Félix Gomes da Costa, not only for the trust, understanding and support, but also for all the advice he gave me during the work.

To Luís Caires, Margarida Mamede and remaining colleagues for their incentive.

To Prof. João Bento, for his great support and trust.

To my friend Miguel Afonso, you'll never be forgot.

To all my friends, in particular to Miguel, Francisco, Nuno, Tiago, Paleta and David, for being there when I needed them.

To my *children* at Faculdade de Ciências, for their support and friendship.

To David Tibet, for the never ending source of inspiration.

To my parents and brother, for everything.

Contents

Chapter 1

Introduction

> Observe constantly that all things take place by change, and accustom thyself to consider that the nature of the Universe loves nothing so much as to change the things which are, and to make new things like them.
>
> *Marcus Aurelius, Meditations*, IV

Biological studies have always constitute a large pool of inspiration for the design of engineering systems. These last decades, two biological systems have provided a remarkable source of inspiration for the development of new types of algorithms: they are neural networks [Rojas, 1996] and evolutionary algorithms [Goldberg, 1989].

In recent years, another biological inspired system has attracted the attention of researchers, the natural immune system and its powerful information processing capabilities [Maçãs, 1997, Martins, 2000, Tarakanov et al., 2003, de Castro and Timmis, 2002]. In particular, it performs many complex computations in a highly parallel and distributed fashion. The key features of the immune system which provide several important aspects to the field of information processing are: recognition, feature extraction, diversity, learning, memory, self-regulation, distributed detection, probabilistic detection, adaptability, specificity, etc.

It is to be noted that the mechanisms of the immune system are remarkably complex and poorly understood, even by immunologists. Several theories and mathematical models have been proposed to explain the immunological phenomena [Perelson and Weisbuch, 1997, Lord, 2002]. There

is also a growing number of computer models to simulate various components of the immune system and the overall behavior from the biological point of view [Celada and Seiden, 1992, DasGupta, 1998]. Those approaches include differential-equation models, stochastic-equation models, cellular-automata models, shape-space models, etc.

The models based on immune system principles, such as the clonal selection theory [Burnet, 1959, Ada and Nossal, 1987, Forsdyke, 1995], the immune network model [Jerne, 1974, Coutinho, 1995] or the negative selection algorithm [Jerne, 1955], have been finding increasing applications in fields of science and engineering [DasGupta, 1998], such as: computer security, virus detection, process monitoring, fault diagnosis, pattern recognition, etc.

Although the number of specific applications confirms the interest and the capabilities of this principles, the lack of a general purpose algorithm for solving problems based on them contrasts with the major achievements in that area for other Biologically inspired models.

In this thesis we will propose an algorithm capable of problem solving. Our intent is to use this algorithm much in the same way genetic algorithms and neural networks are used to solve problems.

Our work focused on describing the algorithm itself and how to use it, on characterizing it as a Probabilistic Algorithm, on providing a set of examples and on comparing it to other models algorithms and discuss ways of comparing them.

The next four chapters take each of these items in turn. The last chapter concludes with a summary of the thesis and discussion of future work. Sample code is provided in the appendix.

Contents of the thesis:

Chapter 2 – In this chapter we present an Immunological based Algorithm, inspired by the hypothesis of clonal selection theory and idiotypic network theory, capable of solving problems. Besides presenting the algorithm itself we briefly discuss his various parameters. It is also discussed how to encode and present a problem to the algorithm and how to extract results from its outcome.

Chapter 3 – In this chapter we outline a probabilistic analysis of the Immunological Algorithm to gain a better understanding of its dynamics. It is presented a formal description of the problem and algorithm, and then analyze the distribution of the various steps of the process.

Chapter 4 – In this chapter we present some examples of how to use The Immunological Algorithm to solve problems. These problems are also solved using other types of biologically inspired solving problems models. It is also briefly discussed the comparative results of the various algorithms.

Chapter 5 – In this chapter we discuss several metrics which could help measuring the relative success of the various selectionist systems. These metrics concern not only the quality of the solutions obtained, but also the efficiency and the adequacy of the systems to the different problems. Finally, the examples used on the previous chapter are revisited using the metrics proposed.

Appendix A – In this chapter we present a set of classes/interfaces programmed in Java that provide a framework for the creation and use of Immunological Algorithms.

Chapter 2

The Immunological Algorithm

> Solomon saith: *There is no new thing upon the earth.* So that as Plato had an imagination, *that all knowledge was but remembrance;* so Solomon given his sentence, *that all novelty is but oblivion.*
>
> *Francis Bacon, Essays*, LVIII

In this chapter we present an Immunological based Algorithm, inspired by the hypothesis of clonal selection theory and idiotypic network theory, capable of solving problems. Besides presenting the algorithm itself we briefly discuss his various parameters. It is also discussed how to encode and present a problem to the algorithm and how to extract results from its outcome.

2.1 The Biologic Model

The natural immune system is a very complex system with several functional components [Paul, 1993, Herbert et al., 1995]. It is constituted by a network of interacting cells and molecules which recognizes foreign substances and marks them for later destruction. These foreign substances are called *antigens*. The molecules of the immune system that recognize antigens are called *antibodies*. An antibody does not recognize an antigen as a whole object. Instead, it recognizes small regions called *epitopes*. An antibody recognizes an antigen if it binds to one of its epitopes. The binding region of an antibody is called the *paratope*. The strength and the specificity of the interaction between antibody and antigen is measured by the affinity of

the interaction. The affinity depends on the degree of complementarity in shape between the interacting regions of the antibody and the antigen. A given antibody can typically recognize a range of different epitopes, and a given epitope can be recognized by different antibody types. Not only do antibodies recognize antigens but they also recognize other antibodies if they have the right epitope. An epitope characteristic for a given antobody type is called an *idiotope*. Antibodies are produced by cells called *B-lymphocytes*. B-lymphocytes differ in the antibodies that they produce. Each type of antibody is produced by a corresponding lymphocyte which produces only this type of antibody. When an antibody on the surface of a lymphocyte binds another molecule (antigen or other antibody), the lymphocyte is stimulated to clone and to secrete free antibodies. The molecule to which an antibody is bound is marked for destruction. The destruction is done by other cells in the immune system. In contrast, lymphocytes that are not stimulated die after days. Thus, a selection process is at work here where those antibodies that are stimulated by antigens or antibodies are amplified, while the other antibodies die out.

In its simplest terms, the task of the immune system is to identify those things that are *self* (that naturally belong in the body) and those that are *non-self* (foreign or otherwise harmful material). The non-self substances are further categorized in order to induce an appropriate type of defensive mechanism. The immune system learns through evolution to distinguish between foreign antigens and body's own cells. There exist several theories to explain immunological phenomena and computer models to simulate various components of the immune system. However, the natural immune system is also a source of inspiration for the developing intelligent methodologies towards problem solving. These methodologies focus on the high parallel and distributed aspects of the immune system. The key features they try to emulate are: learning, memory, and associative retrieval to solve recognition and classification tasks.

One of the most popular theories among researchers is the *immune network model* proposed by Niels Jerne [Jerne, 1974]. Jerne hypothesized that the immune system is a regulated network of molecules and cells that recognize one another even in the absence of antigen. Such networks are often called *idiotypic networks* which present a mathematical framework to illustrate the behavior of the immune system. His theory is modeled with a system of differential equations which simulates the dynamics of lymphocytes, the increase or decrease of the concentration of lymphocyte clones and the corresponding immunoglobins. The idiotypic network hypothesis is based on the concept that lymphocytes are not isolated, but communicate with each other through interaction among antibodies. Jerne suggested that

during an immune response antigens will induce the creation of a first set of antibodies. These antibodies would then act as antigens and induce a second set of *anti-idiotypic* antibodies. The repetition of this process produces a network of lymphocytes that recognize one another. With this hypothesis Jerne explains the display of memory by the immune system.

Other important issues are the incorporation of new types in the immune network and the removal of old types. The autoregulation of the immune system keeps the total number of different types roughly constant. Although new types are created continously, the probability that a newly created type is incorporated in the immune system is different for different types. In natural evolution, the creation of new individuals result from the mutation and crossover of the individuals in the population. In the immune system, new antibody types are created preferably in the neighbourhood of the existing high-fit antibody types. The biased incorporation into the immune network of randomly created species is called the *recruitment strategy* [Bersini and Varela, 1991, Bersini, 1992, Bersini and Seront, 1993].

Also fundamental for the present formulation is clonal selection theory [Burnet, 1959, Ada and Nossal, 1987, Forsdyke, 1995]. This theory states that there is a biunivocal relation between lymphocytes and receptors, hence each kind of lymphocyte has only one kind of receptor. To explain the predominance or the disappearing of certain kinds of lymphocytes, the theory assumes a highly diverse initial population, composed by randomly distributed lymphocytes. The population evolves though the selection of the most well adapted individuals. The recognition of antigens produces a reaction of lymphocytes are stimulating their reproduction, through the production of clones. The measure of the adaptability of a lymphocytes, is proportional to the intensity of the reaction, giving more chances of survival to those kinds of lymphocytes more stimulated on a given environment. This natural selection process together with the high rate of mutation found on the reproduction stage, conducts to a increase of the concentration of the lymphocytes with the highest molecular affinity with the form they try to eliminate.

It is also interesting to compare this model with other biologically inspired models, such as the model for natural evolution of species (in genetic algorithms) and the model neuronal group selection (in neural networks) [Holland, 1975, Paton, 1994]. The main similarities are: each system consists of a population of units; all systems display a high degree of diversity; the units of the population are tested by the environment; due to differences between units, these units respond differently to their environment and the fitter units of the population are selected and amplified; the diversity within the population is result of an autonomous process not influenced by the envi-

ronment; when ideal functioning are apparently insensitive to small damages to its components and structure; all systems are regulated by feedback mechanisms; they all share the property that a perfect recognition is not necessary for the triggering of a response; etc. Other aspects that can be taking into account when comparing the three models are: anatomy, structure, interconnectivity, response to stimuli, maturation, adaptation, parallel processing capabilities, memory, contextual recognition, approximate recognition, robustness, feedback mechanisms, etc.

2.2 The Algorithm

The algorithm presented here follows closely the model proposed by Farmer *et al* [Farmer et al., 1986] for the dynamics of a system based on the idiotypic network model. The dynamics of the model is described using a set of differential equations of the form:

$$\dot{x}_i = c[\sum_{j=1}^{N} m_{ji}x_j x_i - k_1 \sum_{j=1}^{N} m_{ij}x_i x_j + \sum_{j=1}^{n} m_{ji}x_i y_j] - k_2 x_i \qquad (2.1)$$

Letting there be N antibody types, with concentrations $\{x_1,x_2,\ldots,x_N\}$, and n antigens with concentrations $\{y_1,y_2,\ldots,y_n\}$. Let also m_{ji} there be the affinity between the paratope of the antibody i and the epitope of the antigen j.

The first term represents the stimulation of the paratope of an antibody of type i by the epitope of an antibody of type j. The second term represents the suppression of the antibody of type i when its epitope is recognized by the paratope of type j. The constant k_1 represents the possible inequality between stimulation and suppression. The third term models the stimulation provided by the recognition of the antigen j (with concentration y_j) by the antibody of type i. The form of these terms is dictated by the fact that the probability of a collision between an antibody of type i and an antibody (or antigen) of type j is proportional to $x_i x_j$ (or $x_i y_j$). The final term models the tendency of cells to die in the absence of any interactions, at a rate determined by k_2. The parameter c is a rate constant that depends on the number of collisions per unit of time and the rate of antibody production stimulated by a collision.

Since we aim to model, with the present algorithm, the evolution of a population of possible solutions to a problem, and since we are mostly corresponding to the antigenic population. We now get the following set of

equations:

$$\dot{x}_i = c[\sum_{j=1}^{N} m_{ji}x_j x_i - k_1 \sum_{j=1}^{N} m_{ij}x_i x_j] - k_2 x_i \tag{2.2}$$

With some further simplifications we get the equation:

$$\dot{x}_i = [c\sum_{j=1}^{N} x_j(m_{ji} - k_1 m_{ij}) - k_2]x_i \tag{2.3}$$

and finally:

$$\dot{x}_i = (c\sum_{j=1}^{N} n_{ij}x_j)x_i \tag{2.4}$$

where $n_{ij} = m_{ji} - k_1 m_{ij} - \frac{k_2}{cN}$.

Since the system of differential equations is not integrable using algebraic processes, the algorithm uses the discretization of this set of differential equations as an update rule for the concentrations of the various antibody types. To the numerical integration we used the Euler method, which consists of a linear polynomial approximation to each $x_i(t)$ through the Taylor series until the first derivative. Therefore, the concentration of each antibody is approximated, on each step of the simulation, using the update rule:

$$x_i(t+h) = x_i(t) + hf(x(t))$$

Since we can consider the time of the simulation discrete and the concentration of all antibody types is calculated at each instant, we can use a unity as h. Hence, the variation of the concentration of each antibody type between instants t and $t+1$ is approximated by the equation:

$$x_i(t+1) = x_i(t) + x_i(t)\ \left(c\sum_{j=1}^{N} n_{ij}x_j(t)\right) \tag{2.5}$$

If $k_1 = 1$ (equal stimulation and suppression) and $k_2 > 0$ then every antibody type eventually dies due to the damping term. Letting $k_1 < 1$ favors the formation of reaction loops, since all the numbers of a loop can gain concentration and thereby fight the damping term. As N increases, so does the length and number of loops. The robust properties of the loops allow the system to remember certain states even when the system is disturbed by the introduction of new types.

Along with the three parameters (c, k_1 and k_2), which allow the tunning of the algorithm, two others are necessary for the algorithm definition. We need to set a *death threshold* and a *recruitment threshold*. The first establish

the minimum concentration under what a given antibody type is eliminated from the population. The second indicates the minimum concentration above what a given antibody is sufficiently stimulated to initiate the recruitment process. Other parameters might be necessary depending on the recruitment process used on a given implementation.

We can now formulate the Immunological Algorithm as the following steps:

```
Randomly initialize initial population
Until termination condition is met do
    Update concentrations using equation formula
    Eliminate antibodies under death threshold
    Recruitment of new antibodies
```

As usual with this kind of algorithms, several types of termination conditions can be considered (time, number of generations, stabilized population, maximum concentration,...). As for the recruitment process, there are also many approaches that can be followed. The simplest one would be just clone and mutate the antibodies with a probability proportional to their concentrations, but we can use a scheme where crossover and other evolutionary processes are involved.

2.3 Problems and Results

In order to use the Immunological Algorithm to solve a problem we need to know how to encode the problem and how to extract the result once the algorithm stops.

The first thing we need to establish is the representation of the problem. Each antibody represents a possible solution to the problem. Although the usual binary representation is always possible, sometimes it is preferable to use a different representation in order to avoid invalid solutions.

The second, and probably most important, thing we have to setup is the function that computes the affinity between one paratope and an epitope. For this function to be effective it must take into account not only the affinity between the two antibodies but also enhance this value according to the relative quality of the recognizer paratope as a possible solution to the problem. This function corresponds to the m_{ji} factor of differential equations describing the dynamics of the system. Given any two antibodies this function computes the affinity between them.

A third thing that need to be established is the method of recruitment used by the algorithm. The method can be as simple as clone and mutate antibodies with probability proportional to their concentrations, but it can also include other methods such as different types of crossover or any other evolutionary method.

Assuming that only clone and mutation are used as a method of recruitment, we need also to program the mutation function which given an antibody mutates producing a new possible solution of the problem. This function is called when a given clone is selected. If the newly generated antibody doesn't exist in the population it is introduced with the fixed concentration. This function is the responsible for the introduction of diversity of the algorithm.

The simplest implementation of the algorithm uses a vector of antibodies (with their corresponding concentration). On a first stage, it updated the concentrations of the antibodies using the update rule. On a second stage, it updated the vector entries removing and inserting antibodies according to the threshold values defined.

As we might expect several possible encodings exist for a given problem and the choice will greatly influence the results obtained. Besides the various algorithm parameters other elements that can influence the results obtained are the initial population (although the algorithm states that it is randomly generated we can consider the situation where the initial population is carefully chosen), the dimension of the initial population (N), etc.

Another procedure that proved to be useful for the stability of the algorithm is the normalization of the concentrations, so that the total concentration of the antibodies would sum one.

Considering the final outcome of the algorithm we consider the solution to the problem found by the process to be the individual with the highest concentration of the population. Since the concentration is closely related with the probability of using a given antibody it seems reasonable to take as answer to problem the most probable answer found. Several other schemes could be considered depending of the problem itself.

Chapter 3

The Probabilistic Computational Model

> This web of time – the strands of which approach one another, bifurcate, intersect or ignore each other through the centuries – embrace every possibility.
>
> *Jorge Luis Borges, The Garden of Forking Paths*

In this chapter we outline a probabilistic analysis of the Immunological Algorithm to gain a better understanding of its dynamics. It is presented a formal description of the problem and algorithm, and then analyze the distribution of the various steps of the process.

3.1 Probabilistic Algorithms

In mathematics, a probabilistic algorithm is an algorithm that with very high probability gives a correct answer, but not with certainty. There are a number of discrete problems for which only an exact result is acceptable and where the introduction of randomness influences only on the ease and efficiency in finding the solution. For such problems where trivial exhaustive search is not feasible probabilistic algorithms can be applied giving a result that is correct with a probability less than one. The probability of failure can be made arbitrary small by repeated applications of the algorithm. One incentive for using probabilistic algorithms is that their application does not normally require sophisticated mathematical knowledge. Further, the pro-

gramming is often rather trivial which means that an acceptable approximation can be obtained quickly. One can say that the use of probabilistic algorithms sometimes allow that theoretical knowledge and analytical work is compensated for by making extensive simple machine computations. In some other cases the probabilistic algorithms are the simplest and even the most efficient available and for some problems no other feasible algorithm is known to exist.

We can define a probabilistic algorithm as a procedure that behaves in a deterministic way, except that occasionally it takes decisions with a fixed probability distribution [Balcazar et al., 1995].

Our probabilistic model of computation is very similar to a nondeterministic model. The essential difference is that instead of "guessing" the next move, we "toss-a-coin" and make a decision as a function of the outcome.

Intuitively it is easy to recognize the Immunological Algorithm to be a probabilistic algorithm. Considering the algorithm described in the previous chapter, the factors of uncertainty are the randomly generated initial population, the selection of antibodies for recruitment and the generation of new antibodies. Each of these factors can be described in a probabilistic form.

In the next two sections we will address the problem of describing the Immunological Algorithm in a probabilistic form.

3.2 Problem Definition

We have a population of N antibodies. Each antibody is represented as an n item sequence. In generation k, the item j of the antibody i, is a random variable $y_{ij}^k \in Y_j$, where Y_j is the finite set of items that the item j may take. Assume that the cardinality of Y_j is a, independent of j. If the cardinalities differ, let $a = \max_j |Y_j|$ and pad other items with dummy values. The concentration of antibody i of generation k is the random variable x_i^k, with $0 \leq x_i^k < 1$. Assuming the normalization previously referred we have $\sum_{i=1}^{N} x_i^k = 1$.

Assumption 1 *For the purpose of the following analysis, we assume a unique optimal solution so that each item of the solution antibody has an unique correct value.*

Assumption 2 *In order to develop results with some generality, we assume a surrogate affinity measure that is proportional to the antibody quality. The quality is measured by the number of correct values in the antibody sequence.*

Although these assumptions are not always true, since many problems have multiple optima and the affinity measure can take any form, they are reasonable in most cases and allow an analysis of the algorithm that hopefully will lead to insights that generalize beyond the assumptions.

By Assumption 1, there is a single correct value for each item in the solution antibody. Let b_{ij}^k be the indicator random variable of the event that y_{ij}^k has the correct value. Let $S_i^k = \sum_{j=1}^n b_{ij}^k$ be the number of correct values in antibody i of the generation k. By Assumption 2, the affinity function will be proportional to S_i^k (we will assume that the most significative part of m_{ij} is in fact S_i^k and write $m_{ij} \approx S_i^k$). The population average in generation k is defined as $G^k = \frac{1}{m}\sum_{i=1}^N S_i^k$. The best element of the population is $A^k = \max_i S_i^k$. Given a generation, $\{S_i^{k-1}\}_{i=1}^N$, we can construct the next generation by successively updating and normalizing antibody concentrations, deleting the antibodies under the *death threshold* and incorporating newly generated antibodies.

The Immunological Algorithm to be studied can be formally stated:

(Initialize) Generate y_{ij}^0 uniformly from {1,2,...,a} for $i = 1, \ldots, N$; $j = 1, \ldots, n$. Assign x_i^0 to its initial value. Calculate all S_i^0 and $A^0 = \max_i S_i^0$. $k \leftarrow 1$.

(Iterate) While $A^{k-1} < n$ do

(Update) Use the rule $x_i^k = x_i^{k-1} + x_i^{k-1}(c\sum_{j=1}^N n_{ij}x_j^{k-1})$ to update the concentrations of the antibodies. Normalize concentrations.

(Eliminate) Delete all antibodies with concentration x_i^k inferior to the *death threshold*. Update the value of N.

(Recruitment) Randomly select, with probability x_i^k, for recruitment antibodies with concentration superior to the *recruitment threshold*. Mutate the selected antibodies (modifying item $l \in \{1, \ldots, n\}$ to a value from {1,2,...,a}) introducing the newly created antibodies to the population. Update the value of N.

(Evaluate) Evaluate each S_i^k. Let $A^k = \max_i S_i^k$.

(Increment) $k \leftarrow k + 1$.

(end iteration)

The goal of this chapter is to derive the distribution of A^k and the population average, G^k.

3.3 Derivation of the Distribution

In Section 3.3.1 we will analyze the initialization step of the algorithm. Section 3.3.2 models the recruitment process for selected antibodies with concentration superior to the *recruitment threshold*. Independence of offspring is discussed leading to a distribution of the best element of a generation given the distribution of the previous generation.

The remaining steps of the Immunological Algorithm display a behavior completely deterministic. The first one, *Update*, will only affect the distribution of concentrations between the antibodies of the population. This will only influence the choose of an antibody for delection and for recruitment. The second one, *Eliminate* will affect the population, but since the antibodies eligible for recruitment could never be eligible for delection it will not affect the *Recruitment* step. We will then center our attention in the two steps that display probabilistic behavior.

3.3.1 Initialization

For the initial generation, y_{ij}^0 is taken uniformly from the set of potential items, represented by $\{1,2,\ldots,a\}$. Recalling that exactly one of these items is correct, the probability of randomly choosing the correct item is $p = 1/a$. Hence, b_{ij}^0 is a Bernoulli random variable with parameter p. By construction, all y_{ij}^0 and b_{ij}^0 are independent random variables.

Then S_i^0 has a binomial distribution with parameters (n,p). Further, all S_i^0 are independent from each other, though clearly not independent of the corresponding y_{ij}^0 and b_{ij}^0. The initial population affinity average is $E[S_i^0] = np = n/a$.

With N independent members of the population, we have:

$$P(A^0 \leq J) = P(\max_i S_i^0 \leq J) = P(S_i^0 \leq J, \forall i) = [F(J)]^N \equiv F^N(J)$$

where $F(J)$ is the cumulative distribution function of a binomial (n,p) random variable. The expectation of A^0 can be found easily by the expression $E[A^0] = \sum_{J=0}^{n-1}[1 - F^N(J)]$.

3.3.2 Recruitment

In generation 0, members of the population are independent by construction making such analysis simple. Subsequent generations depend from the previous ones. However, their correlation is greatly reduced given the distribution of affinity values in population $k-1$ and sufficiently large N.

The first part of the recruitment process is the selection of antibodies. This selection process is determined by the value of antibody concentrations. Only the ones with concentration superior to the *recruitment threshold* are eligible for reproduction. Without loss of generality, for the remainder of this section, we will consider only those antibodies.

The process of *Recruitment* will maintain the population with the possible inclusion of newly created antibodies resulting from a process of mutation from the selected ones.

Given an antibody selected at random, indexed p_1, with affinity $S_{p_1}^{k-1} = J_1$ we derive the distribution of the affinity of the offspring, S_t^k, $t > N$ (since we are introducing a new antibody and increasing the size of the population). To facilitate this analysis we make one assumption.

Assumption 3 *Given that the parent have J_1 correct items, the location of the correct items is distributed uniformly with the parent's antibody and are independent of each other.*

Assumption 3 will be true in early generations, but becomes less accurate as the population converges. The impact of this assumption will be judged empirically.

Given the selected antibody, the mutated item $l \in \{1, \ldots, n\}$ will either change from an incorrect value to a correct one, or change from an incorrect value to another incorrect one, or change from a correct to an incorrect value.

Then for the offspring we have

$$S_t^k = \sum_{j \neq l}^{n} b_{tj}^k + w_l$$

where w_l is a Bernoulli random variable with parameter $1/a$.

The maximum value of J can take is given when all the correct items remains unchanged and the item selected mutates to a correct value. In this situation we have $S_t^k = S_t^{k-1} + 1$.

Under the Assumption 3 we have

$$P(S_i^k = J | S_{p_1}^{k-1} = J_1) = \binom{n}{J} J_1^J (1 - J_1)^{n-J} \tag{3.1}$$

For a sufficiently large N, we may approximate

$$P(S_i^k \leq J_i, i = 1, 2, \ldots, N) \approx \prod_{i=1}^{N} P(S_i^k \leq J_i) \tag{3.2}$$

we can now approximate the important values for each generation.

Conditioning on parentage, we have

$$P(S_i^k \leq J) \approx \sum_{J_1=0}^{n} P(S_i^k \leq J | S_{p_1}^{k-1} = J_1) P(S_{p_1}^{k-1} = J_1) \tag{3.3}$$

The first factor is known from (3.1). The last factor is known from an inductive hypothesis. Then $E[S_i^k] = \sum_{J=0}^{n} [1 - P(S_i^k \leq J)]$. By the approximation (3.2), $P(A^k \leq J) \approx [P(S_i^k \leq J)]^m$.

Finally $E[A^k] \approx \sum_{J=0}^{n-1} [1 - P(A^k \leq J)]$. To compute $E[A^k]$, we compute the anchor distribution of S_i^0 as in Section 3.3.1, then inductively compute the distributions of S_i^k via (3.3).

Chapter 4

Examples and Comparisons

> All human errors are impatience, a premature breaking off of methodical procedure, an apparent fencing-in of what is apparently at issue.
>
> *Frank Kafka, The Collected Aphorisms*

In this chapter we present some examples of how to use The Immunological Algorithm to solve problems. These problems are also solved using other types of biologically inspired solving problems models. It is also briefly discussed the comparative results of the various algorithms.

4.1 Simple Function Optimization

The first example selected is a classical maximization problem. The aim of this example is to show how to use the Immunological Algorithm in its simplest form to solve a problem that is very well studied for other Biologically inspired algorithms.

The selected function was the power function $f(x) = (x/c)^n$, where c has been chosen to normalize x, and n has been chosen as 10. This function was selected both for its simplicity and for relative hardness compared to other power functions ($f(x) = x^2$ for example).

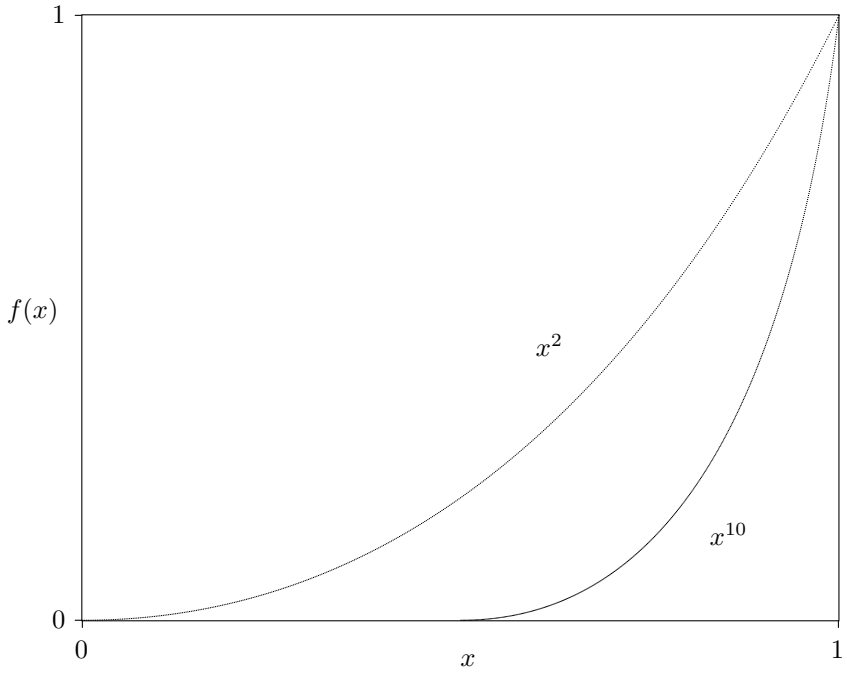

Fig. 3.1 - Comparison of the functions x^2 and x^{10} on the unity interval.

As presented in Chapter 2, the first thing we needed to do in order to use the Immunological Algorithm is to establish the representation of the problem. For this first example it was selected the simplest representation possible, a binary representation, with 30 bits per antibody corresponding to a large search space ($2^{30} \approx 1.07 \times 10^{10}$).

The second thing we needed to define is an affinity function. As suggested when the algorithm was presented, we considered two components to compute this function. The first component represents the quality of the antibody, given by the function f we aim to optimize. The second component represents the correlation between an antibody and any other antibody of the population, for this we used the Hamming distance between the two antibody strings.

The Hamming Distance can be defined as a measure of the difference between two messages, each consisting of a finite string of characters, expressed by the number of characters that need to be changed to obtain one from the other.

Given two antibodies (u and v) of size m the Hamming distance between the two can be computed by the function:

$$\mathcal{H}(u,v) = \sum_{i=1}^{m} u_i \left(1 - v_i\right) + \sum_{i=1}^{m} v_i \left(1 - u_i\right) \tag{4.1}$$

The final affinity function results from the product between the two components described:

$$\mathit{Affinity}(u, v) = f(u) \times \mathcal{H}(u, v) \tag{4.2}$$

The third element to select was the recruitment method. As in the previous choices, we selected the simplest method possible. We chosen to add to the population mutated clones of the antibodies with concentration above the *recruitment threshold*.

The algorithm was used with several initial population sizes and various algorithm parameters and, most of the times, it was capable of finding a near optimal solution in only a few generations.

4.2 The Iterated Prisoner's Dilemma

For this second example we chosen the Iterated Prisoner's Dilemma. The aim of this example is to show the use of the algorithm with a less trivial problem, and the use of more sophisticated antibody representation and affinity function.

The prisoner's dilemma is a classic problem of conflict and cooperation [Poundstone, 1993, Axelrod, 1997]. In its simplest form each of two players has a choice of cooperating with the other or defecting. Depending on the two players' decisions, each receives payoff according to a payoff matrix.

When both players cooperate they are both rewarded at an equal, intermediate level (reward, R). When only one player defects, he receives the highest level of payoff (temptation, T), while the other player gets the sucker's just deserts (sucker, S). When both players defect they each receive an intermediate (penalty, P).

The prisoner's dilemma has often been cited as a simple yet realistic model of the inherent difficulty of achieving cooperative behaviour when rewards are available for the successful miscreant. The problem is called the prisoner's dilemma because it is an abstraction of the situation felt by a prisoner who can either cut a deal with the prosecutor and thereby rat on his partner in crime (defect) or keep silent and therefore tell nothing of the misdeed (cooperate).

The problem is made more interesting by playing it repeatedly with the same group of players, thereby permitting partial time histories of behaviour to guide future decisions. This so-called iterated prisoner's dilemma has drawn interest from game theorists for a while.

		Player 2	
		Cooperate	Defect
Player 1	Cooperate	R=6, R=6	S=0, T=10
	Defect	T=10, S=0	P=2, P=2

Table 3.1 - Prisoner's Dilemma Payoff Matrix.

A strategy for the Iterated Prisoner's Dilemma is not obvious. The goal to the algorithm was to come up with the best possible strategy.

As ever, our first concern is to provide a representation for the possible solutions of the problem. To represent a strategy we will need to what to do next based on the history of past plays.

In our representation, to decide on a current move, the current player will look at the past three rounds before making a decision. If a single round can be represented by 2 bits: CC, CD, DC, DD, a three round memory would require 6 bits. By choosing to represent a cooperation (C) as a 0 and a defection as a 1, a simple history pattern can be recorded. For instance, 10 00 01 will represent that Player 2 defected three rounds ago, they both cooperated two rounds ago, and Player 1 defected last round.

In order to choose a current move, a Player must have a strategy for all possible previous three rounds. In other words, a Player must have 64 (2^6) different strategies on how to play. Also, a strategy consists of defecting or cooperating, which can be represented by a single bit (0 for cooperate and one for defect).

These facts allow the creation of our bit string for the algorithm. Each bit represents a strategy for a given past three rounds. The index value for the bit string is directly computable from the 6-bit history, it is merely the conversion of the 6-bit history into the corresponding number.

So, the representation of a strategy for the prisoner's dilemma is a 70-bit string. The first 64-bits are to be used to determine the current move based on past moves. The last 6 bits are to be used to determine the first move.

Of course this representation can be generalized to consider the n previous rounds to make the decision, instead of just three. Note however that the size of the antibodies will increase exponentially (antibody size for n rounds would be $2^{2^n} + 2^n$).

Our second concern is to choose an affinity function. The obvious choice is to make different strategies play against each other and use the difference of the scores as the affinity. Obviously, the number of rounds we choose to play will influence the accuracy of the affinity function. On the other hand, since the algorithm will test the affinity of each antibody against all other, choosing a large number of rounds will slow down the algorithm greatly. For this example we used and affinity function that will play 20 rounds.

Our last concern is to provide a recruitment method for the algorithm. As we did on the first example, as recruitment method we use cloning and mutation. A mutation function can easily be implemented by randomly change the value of a small number of bits.

Finally, as a way to improve the performance of the algorithm, we introduced in the initial population several antibodies representing well known proposed strategies, such as:

- nice – Player cooperates all the time;
- mean – Player defect all the time;
- tit-for-tat – Player cooperates for the first move, and the copies the other player's last move;
- opp-tit-for-tat – Player defects for the first move, and the copies the other player's last move;
- cd – Periodic behavior - plays CDCDCD
- ccd – Periodic behavior - plays CCDCCD
- dcc – Periodic behavior - plays DDCDDC
- pavlov – Player cooperates if both players played the same action in the last round.

Even with a random initial population the algorithm was able to discover strategies that beat the overall performance of the tit-for-tat strategy.

As a final remark, we must say that the algorithm seemed specially well suited for this kind of problem of truely competing solutions.

4.3 The N-Queens problem

The final example is the well know N-Queens problem. The aim of this example is to show how we can solve the same problem with different Biologically inspired solving problems algorithms and how these approaches relate to each other.

The N-Queens problem consists to place N queens on an NxN chessboard, so that no two queens attack each other. An alternate way of expressing the problem is to place N *pieces* on a NxN grid such that none of them shares a common row, column, or diagonal. The N-Queens problem is one of NP-hard problems and has been used as a testbed for the development and benchmarking of search algorithms.

As previously, the first thing to establish is the representation of the problem. First, we considered the most direct approach, that is, representing the chessboard as an NxN bitstring, with each bit representing a square, a one indicating the presence of a queen, and a zero indicating it's absence. This representation had some clear problems, however. First, the antibody length required was fairly large, and second, it seemed very likely that states where more than one queen or no queen is positioned per row, or per column. To solve these problems we propose an alternative representation of the problem.

First recall that the N-Queens problem requires us to place n queens on a NxN grid such that there are no attacks. We know that the naive approach to the search space creates too many solutions to search. So we formulate each solution as an n-tuple (q_1, q_2, q_3,..., q_n), where the i^{th} element represents in the i^{th} row, and the q_i^{th} value represents in the q_i column of that row. Formatted in this way we can eliminate all solutions that contain the same value in more than one position, this cuts us down to $n!$ solutions. Which is much better but still too many solutions to solve for given a large value of n. This representation can still represent all possible valid solutions, because all valid solutions require that no two queens can share a row or column (or they would be able to attack each other).

In this case a solution is represented by a permutation (q_1,..., q_n) of numbers (1,2,...,n). A problem arises when trying to generate random permutations, often duplicating some numbers or missing others. Other problems are related with the use of the mutation recruitment operator to produce diversity after the clonal threshold is reached. To solve these problems we used signatures instead of permutations.

A signature is a sequence of numbers that represent a permutation, that are not themselves a permutation. A signature $S = \langle s_1, s_2, \ldots, s_N \rangle$ satisfies the property $1 \leq s_i \leq N - i + 1$ for all i. S specifies that 1 is in position s_1, 2 is in position s_2 of the remaining $N - 1$ positions, 3 is in position s_3 of the $N - 2$ remaining positions, and so forth.

The advantages of signatures presented are:

- Signatures are easy to generate;
- Signatures can be easily mutated (regenerate one of the entries);
- There exist straightforward algorithms to convert signatures into corresponding permutations

The second, and fulcral, issue to be considered in order to use the Immunological Algorithm to the solving of the N-Queens problem is the choice of an affinity function. Again, this affinity function must take into account two

aspects, first the quality of a given antibody as possible solution of the problem, and second the capacity to recognize (and eliminate) other competing possibilities.

In the quality issue, we must recognize that our representation will only have to test whether two queens at positions (i,j) and (k,l) are on the same diagonal. We can test it by some observations:

- Every element on the same diagonal that runs from upper left to lower right has the same row and column value or same (row-column) value;
- Every element on the same diagonal that goes from the upper right to the lower left has same row+column value.

So two queens are on the same diagonal if $i-j = k-l$ or $i+j = k+l$. These two equations can be rewritten as $j-l = i-k$ and $j-l = k-i$. That is, if $|j-l| = |i-k|$ then the two queens share the same diagonal.

Since we have already eliminated the row and column conflicts by our formation of the n-tuple. Our only conflict is along the diagonals, so we have devised a scheme to count the number of safe queens as a means to rank candidate solutions.

As second part of the affinity function we chose to use a function which aims to compute the stimulation obtained by recognizing other competing solutions. To do so we used the Hamming distance between the two antibody representations.

Finally, we used an affinity function which is a linear combination of the individual quality function and the recognition stimulation function. Note that both, the number of safe queens and the number of different positions between two antibodies had to be calculated from the permutation denoted by the signature representation.

To complete the formulation of our problem we needed only to provide a recruitment method. Again we used cloning and mutation as recruitment method and as a way to generate diversity. Given our representation of the problem, a mutation function can be easily defined by randomly choosing a position i and then randomly generate a new value for q_i. This new value must still satisfy the $1 \leq q_i \leq N - i + 1$ condition. Although this simple process may change completely the permutation denoted by the antibody, signatures will remain valid and will preserve their convenient properties.

With this formulation the Immunological Algorithm has proved to be able to solve the N-Queens problem even for large values of n.

In order to compare results we also tried other Biologically inspired solving problems models. Our second approach was to use Neural Networks. We used a Hopfield network with the Boltzmann distribution [Rojas, 1996].

A Hopfield network consists of n totally coupled units, that is, each unit is connected to all other units except itself. The network is symmetric because the weight w_{ij} for the connection between unit i and unit j is equal to the height w_{ji} of the connection from unit j to unit i. In the Hopfield model it is assumed that the individual units preserve their individual states until they are selected for a new update. The selection is made randomly and is asynchronous.

The Hopfield model with n units and asynchronous dynamics, which starts from any given network state, is guaranteed to reach a stable state at a local minimum of the energy function:

$$E(x) = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ij}x_i x_j + \sum_{i=1}^{n}\theta_i x_i \tag{4.3}$$

Where w_{ij} represents the weight of the connection between the unit i and unit j, x_i represents the state of the unit i and the thresholds of the units are denoted by θ_i.

We encoded the chessboard by a matrix of n x n units of binary state, where $a_{i,j}$ represents the (i,j) square of the board. We needed to define the energy function so that it would achieve its minimum value at solutions.

The term:

$$H_1 = \sum_{j=1}^{n}[(\sum_{i=1}^{n} a_{i,j}) - 1]^2 \tag{4.4}$$

is equal to zero only if there is exactly one queen in each row and strictly greater then zero otherwise.

Similarly, the term:

$$H_2 = \sum_{i=1}^{n}[(\sum_{j=1}^{n} a_{i,j}) - 1]^2 \tag{4.5}$$

achieves its minimum when there is exactly one queen in each column.

To penalize terms with two or more queens per diagonal we used:

$$H_3 = \sum_{j=1}^{n-1}[(\sum_{i=1}^{n-j+1} a_{i,i+j-1}) - \frac{1}{2}]^2 + \sum_{j=2}^{n-1}[(\sum_{i=j}^{n} a_{i,i-j+1}) - \frac{1}{2}]^2 \tag{4.6}$$

for the *positive* diagonals, and

$$H_4 = \sum_{j=2}^{n}[(\sum_{i=1}^{j} a_{i,j-i+1}) - \frac{1}{2}]^2 + \sum_{j=2}^{n-1}[(\sum_{i=j}^{n} a_{i,n+j-i}) - \frac{1}{2}]^2 \tag{4.7}$$

for the *negative* diagonals.

Finally, the energy function H is defined as a linear combination of H_1, H_2, H_3 and H_4 as follows:

$$H = \alpha(H_1 + H_2) + \beta H_3 + \gamma H_4 \tag{4.8}$$

where $\alpha, \beta, \gamma > 0$. Obviously, as all terms are positive, this function is at its minimum, when the position of queens is optimal.

The weights for the network can be deducted from the following considerations. Let x_1, x_2,..., x_n denote the binary states of a column of n individual units. We aim to find the minimum of

$$E(x_1, \ldots, x_n) = (\sum_{x=1}^{n} x_i - 1)^2$$

This expression can also be written as

$$E(x_1, \ldots, x_n) = \sum_{x=1}^{n} x_i^2 + \sum_{i \neq j}^{n} x_i x_j - 2 \sum_{x=1}^{n} x_i + 1$$

For binary states it holds that $x_i = x_i^2$ and therefore

$$E(x_1, \ldots, x_n) = \sum_{i \neq j}^{n} x_i x_j - \sum_{x=1}^{n} x_i + 1$$

which can be rewritten as

$$E(x_1, \ldots, x_n) = -\frac{1}{2} \sum_{i \neq j}^{n} (-2) x_i x_j + \sum_{x=1}^{n} (-1) x_i + 1$$

This expression is isomorphic to the energy function of the Hopfield network (not considering the constant 1, which works only as *bias* and is irrelevant for the optimization problem).

Considering now the whole matrix instead of just a column, it is easily recognizable that we are in the presence of the sum of n independent functions (one per column). This sum corresponds exactly to the term (4.4). Hence, the connections of each unit to all elements in the same column have the weight -2, all others have the weight zero. All units have the threshold -1.

A similar approach can be followed both for the rows and for the diagonals, corresponding to the terms (4.5), (4.6) and (4.7). The overlapping of these terms provides us the necessary weights, which are set to $w_{ij} = -2$, when unit i is different from the unit j and belongs to the same row, column or diagonal as unit j. Otherwise we set w_{ij} to zero. The thresholds of all units are set to -1.

The computer simulation shows, however, that this simple connection pattern does not always provide a correct solution for the N-Queens problem. The connection weights produce an energy function in which local minima with less than n queens are possible.

In order to improve the results we used a Boltzmann machine, which is a Hopfield network where the state of unit i is updated asynchronously according to the rule

$$x_i = \begin{cases} 1 & \text{with probability } p_i \\ 0 & \text{with probability } 1 - p_i \end{cases}$$

where

$$p_i = \frac{1}{1 + \exp(-(\sum_{j=1}^{n} w_{ij} x_j - \theta_i)/T)}$$

and T is a positive temperature constant.

The difference between a Boltzmann machine and a Hopfield network is the stochastic activation of the units. If T is very small then $p_i \approx 1$ when $\sum_{i=1}^{n} w_{ij} x_j - \theta_i$ is positive. If the net excitation is negative, then $p_i \approx 0$. In this case the dynamics of the Boltzmann machine approximates the dynamics of the Hopfield network and the Boltzmann machine settles on a local minimum of the energy function. If $T > 0$, however, the probability of a transition, or sequence of transitions, from a network state to another is never zero allowing to avoid local minima. Hence we start the process with large values of T proceeding to cool it down to zero.

The results proved to be largely improved by the use of the Boltzmann machine instead of the Hopfield network, providing less non valid solution results.

The third, and final, approach was to use Genetic Algorithms in order to solve the N-Queens problem [Goldberg, 1989].

Much of the discussion about the representation of possible solutions for the N-Queens problem using the Immunological Algorithm is valid for the Genetic Algorithm. We also used signatures to generate permutations. The main concern was to get permutation-respecting Crossover Operators. In order to do so we used the well known *Partially Mapped Crossover* (PMX), *Order Crossover* (OX) and *Cycle Crossover* (CX). We can briefly describe these operators as follows:

- The PMX builds an offspring by taking a sub-sequence from one parent and preserving the order and position of as many values as possible from the other parent;

- The OX takes a sub-sequence from one parent and preserves the relative order of the values from the other parent;

- The CX creates an offspring different from the parents where every position is occupied by a corresponding element from one of the parents.

As mutation operator we invert a segment of the chromossome, given two randomized indexes i and j the chromossome:

$$(q_1, \ldots, q_i, q_{i+1}, \ldots, q_{j-1}, q_j, \ldots, q_n)$$

is mutated to the chromossome:

$$(q_1, \ldots, q_j, q_{j-1}, \ldots, q_{i+1}, q_i, \ldots, q_n)$$

As fitness function we used the same scheme to rank candidate solutions that we used as part of the affinity function in the Immunological Algorithm. Our fitness function checks for diagonal attacks and counts the number of safe queens as the merit of a given chromossome.

However, when creating a new population by crossover and mutation, the best chromosomes may be lost. Hence, elitism is a method which first copies the best chromosomes to the new population. We also used elitism as a mean to rapidly increases the performance of the Genetic Algorithm, by preventing loss of the best-found solution.

All three algorithms were capable of solving the N-Queens problem even for larger values of n.

Chapter 5

Metrics for Result Comparison

> Like every writer, he measured the virtues of other writers by their performance, and asked that they measure him by what he conjectured or planned.
>
> *Jorge Luis Borges, The Secret Miracle*

In this chapter we discuss several metrics which could help measuring the relative success of the various selectionist systems. These metrics concern not only the quality of the solutions obtained, but also the efficiency and the adequacy of the systems to the different problems. Finally, the examples used on the previous chapter are revisited using the metrics proposed.

5.1 Models and Algorithms Comparison

The three biological systems covered in this work have a lot in common but there are also a number of differences. In order to discuss metrics of comparison between them we first focus on the similarities and the differences of the models and the algorithms.

Each system consists of a population of units: a species is a population of genotypes/phenotypes, the immune system is population of B-lymphocytes and the corresponding antibodies and the brain is a population of neural groups [Edelman, 1988].

There is a lot of diversity within the population: no two individuals of a species are identical, about 10^7 antibody types exist in the immune system and the neuronal groups are different in their anatomical structure.

Units of the population are tested by either the external or the internal environment: individual organisms are tested by their niche, antibody types are tested by antigens (external environment) or antibody types in the immune system (internal environment) and neural groups are tested by signals coming from the sense organs (external environment) or by signals coming from other groups (internal environment).

Due to differences between the units, these units respond differently to their environment and the fitter units of the population are selected by the environment and amplified. In the evolution of species, the units of selection are individuals and the better adapted individuals have more offspring the less well-adapted ones. In the immune system, the units of selection are antibodies and the number of activated antibodies increases through the proliferation of the corresponding B-lymphocytes and the release of free antibodies. In the brain, the units of selection are neuronal groups and the groups that respond to an input are amplified through modification of synaptic weights.

The diversity within the population of units is the result of an autonomous process influenced neither directly nor indirectly by the environment. The diversity among the genotypes results from mutation and crossover of the existing genotypes. Mutation and crossover are not affected by the environment. The diversity among the antibody types in the immune system is the result of the recruitment strategy. This strategy is largely unaffected by the external environment: B-lymphocytes activated by an antigen start to mutate at a higher rate than the other B-lymphocytes. This is the only influence of the external environment on the generation of diversity in the immune system. The anatomical differences between neural groups are result of cell movement, cell differentiation and cell death.

Despite of all these similarities there also some of important differences between these three models.

Selectionist principles play a different role in evolutionary theory on the one hand, and the clonal selection theory and the neuronal group selection theory on the other hand. In evolutionary theory, selectionist principles explain how species become more and more adapted to their environment, that is, it explains the evolution of species. However, evolutionary theory says nothing about the mechanisms behind this adaptive behavior. In contrast, it is the functioning of the immune system and the brain that is explained by selectionist principles, that is, these principles clarify the mechanisms resposible for the adaptability of the immune system and the brain. This way, the immune system seems much more important than natural evolution as a metaphor in understanding adaptation in an unknown environment.

Also, adaptation in dynamical systems comes in two types: adjustment and innovation. If adaptation is the result of changes of the parameters in

the system then it is called adjustment. Learning in the neural nets where the weights are the parameters is an example of adjustment. In contrast, if adaptation results from topological changes of the dynamical system then it is called innovation. Adaptation as reflected in the metadynamics of the immune system is an example of innovation. The continuous creation of new antibody types changes the immune system's network topology. In both the evolution of species and the immune system, adaptation is innovation.

On the algorithm level the differences are even more visible. We will discuss next several of the algorithm related factors that differentiate these three approaches.

First we need to take into account the many possible variations of the algorithms themselves. Each of the algorithms can take many forms focusing on different aspects of the models, for instance, different types of neural networks can be considered and different evolutionary algorithms can be used, the use of variants such as elitism, etc. Each of the algorithms is subject to a number of parameters that might influence the final outcome, things like the size and form of the initial population in genetic algorithms, the Immunological Algorithm's parameters or the energy function in a Hopfield network can greatly influence the algorithm's result.

Second, and probably even more important, we have the question of the problem representation used by the algorithms. It is often very difficult to establish a common representation all the algorithms, and even when that is possible to accomplish most of the times this is done by sacrificing better representations, more adequate to the problem in study. Even when the same representation is used, there are also things like which crossover and mutation operator to use in the genetic algorithms or which recruitment strategy to use in the Immunological Algorithm. When different representations are involved it is very hard to differentiate when the high quality of the solution is result of the algorithm or when it is result of a well chosen representation.

Finally, regarding especially the Immunological Algorithm and the genetic algorithm, a factor to consider is the affinity and fitness functions selected. As discussed about the representation, it is often uncertain whether the quality of the outcome of the algorithm is function of its intrinsic quality and adequacy to the problem or function of a carefully selected affinity or fitness functions.

Regarding the differences of the algorithms and the way they might interfere with their final outcome, we strongly feel that the best combination of factors should always be used. Part of the quality of an algorithm is the easiness of its use on a problem and its tunning capacity in order to obtain the best possible outcome. Hence we should always take full advantage of the algorithm's capabilities.

5.2 Metrics

From the discussion of the previous section it is easy to recognize that it is very hard to provide a way to effectively compare the quality of the results, the applicability and the adequacy of the algorithms discussed. Even more if we take into account that we might be dealing with different types of problems.

Concerning the quality of solutions two types of problems can be considered. One where a number of equal quality solutions exist and the algorithm must find one of such solutions. Another where algorithms search though valid solution of variable quality and try to find the best possible solution.

For the first kind of problems it is easy to associate quality with the time needed by the algorithm to provide a valid solution to the problem. One way to compare algorithms on this kind of problems is to make a series of runs of each algorithm measuring the time needed for them to generate a valid solution. The average time needed by each algorithm will give us a measure of its quality. Of course, the greater is the number of runs the most accurate will be this measure.

One example of this kind of problems is the N-Queens problem discussed on the previous chapter. For this problem we made several runs for each of the proposed algorithms and we found that both genetic algorithms and the Immunological Algorithm had results of the same magniture while the neural algorithm proved to be slower.

For the second kind of problems the first thing we need to establish is an objective way to measure the quality of solution, independently of the algorithm who provided it or the actual representation. To do this we propose the definition of a representation capable of holding any possible solution of the problem and a function from that representation to the integral interval [0..1]. Note that this representation does not need to be equal to any of the representations used by the different algorithms to try to solve the problem. What is needed is an effective way to translate from the problem solving representations to the one used by the measuring function.

Again we will make a series of runs of each algorithm using the defined function to measure the quality of the results. The quality assigned to each run will be the value of the function applied to the best solution provided by the algorithm. Note that an alternative to this approach would be to consider as the measure the quality of the algorithm result an average of the values of the final population.

One question that arises is where to stop each run in order to establish the final result and use it as source to the comparison function. Two main criteria were considered: time and number of generations. The first stops the

run after a predetermined time period, while the second runs a fixed number of generations regardless of the time needed. Although both criteria seem equally valid and useful we opted by the use of the first one. The reason for this choice is that frequently time is a serious constraint when dealing with real life problems.

Finally we need to agree on how to interpret the function results from the series of runs. Two alternatives appear as obvious choices: the use of the average of the various function results and the use of the maximum value obtained. Of the two alternatives, the fist one seems more useful as it give a better notion of how we could expect the algorithm to perform. Again, the greater is the number of runs the most accurate will be this measure.

As example of this kind of problems the Simple Function Optimization and the The Iterated Prisoner's Dilemma presented in the previous chapter can be used. For the first example, it is simple to define the comparison function as it coincides with the function we chosen to optimize. For the second example, a possible comparison function would be to test the solution against a set of well known strategies (playing one against another). The result of the comparison function will be the average of the scores normalized by the highest possible score. The number of rounds used by the function should be higher than the used by the affinity function. Since the number of rounds used by the affinity function was 20, we should use no less then 100 rounds to test the quality of the solution.

Another way to survey the quality of an algorithm is to measure the adequacy of the algorithm to a problem or class of problems. This adequacy is even more difficult to measure than the quality of the solutions obtained. This measure largely related with the easiness of selecting all the factor for applying an algorithm to a problem.

It is our conviction that the Immunological Algorithm is specially adequate for problems such as the Iterated Prisoner's Dilemma where possible solutions compete with each other. The same way that genetic algorithms are especially adequate for optimization tasks and neural networks for classification tasks.

Chapter 6

Conclusions and Future Work

> God, I could be bounded in a nutshell and count myself a King of infinite space.
>
> *William Shakespeare, Hamlet*, II

The main contribution of this thesis was to establish a problem solving algorithm based on Farmer *et al* models of the immune system, referred as Immunological Algorithm. The algorithm proved to be capable to solve small to moderate size problems in an efficient and reliable way.

Starting with a brief Biologic Model, we formulated the algorithm itself, its dynamics and its various parameters. It was also discussed how to present problems and how to extract results from its outcome.

Second, we identified the algorithm as belonging to the Probabilistic Algorithms class. To accomplish this, a more formal definition of the algorithm is provided, followed by the analysis of derivation of the distribution of the population concentrations.

Third, we presented several examples of how to use of the algorithm with different problems. With these examples we gave not only an overview of how to use the algorithm, but also a sense of its applicability to different types of problems. Another interesting aspect of the examples was the use of different representations, approaches and the comparison to other Biologically inspired solving problems models. Also related to the examples, we provided an framework for the use of the algorithm in the Java programming language.

Finally, we compared the various Biologically inspired solving problems models discussing several metrics that could help measuring their relative success.

Future work might concentrate on the study of variants of the algorithm and the impact of its various parameters. As with evolutionary algorithms the Immunological Algorithm can be subject of variations, concerning, for instance, the order of operations, different recruitment strategy or inclusion of additional techniques to increase or speed convergency. Also the impact of normalization and the variation of the algorithm's parameters should be object of further study.

Another area for further exploration is the use of the algorithm with a larger set of problems and the study of adequacy of the algorithm for these problems. We might be concerned both with the quality of the results and the study of which class of problems is best suited for the algorithm. The consideration of different representations, affinity functions and recruitment methods dealing with well known problems can prove of particular interest.

Additionally, research could be carried out into the study of parallelization models for the algorithm. A simple and effective embarassingly parallel approach simply consists in running the same algorithm on each processor with possibly different parameters or differently generated initial populations. Also, since the evaluation of the affinity function will often be the most time-consuming part of the computation, then a useful parallel decomposition is to have affinities calculated by different tasks simultaneously. The partition the population into subpopulations that evolve independently with a periodic communication phase between subpopulations can be used to overcome increasingly larger populations during the algorithm's run. The use of these, and other, approaches and its effect on the algorithm's performance should also be subject of attention.

Appendix A

Source Code Framework

> These instincts were given me by Nature.... I am in her hands but a machine which she runs as she likes.
>
> *Marquis De Sade, The 120 Days of Sodom*

In this chapter we present a set of classes/interfaces programmed in Java that provide a framework for the creation of Immunological Algorithms. The main concern in writing this source code was not efficiency but simplicity and extensibility. The objective was, not only to enable an easy understanding of the code, but also to provide a starting point to encode problems (by implementing the two interfaces) and to try different variations of the algorithm (by extending the `ImmunoAlgorithm` class). All the examples presented earlier were developed using this source code as basis. More about these examples can be found on the address `http://ctp.di.fct.unl.pt/~jddp/thesis/`.

A.1 IAntibody interface

```
/**
 * This interface defines the minimum set of methods to be implemented
 * by an Antibody class. These methods provide the basic functionality
 * needed by the Immunological Algorithm. Note that this interface is
 * completely implementation independent.
 *
 * @author      Jose Pacheco
 * @version     %I%, %G%
 */
```

```
public interface IAntibody
{
  /**
    * This function indicates whether some (other) object is "equal to"
    * (this) antibody. Although the Object class defines a default
    * implementation of this method it is usually desirable to define
    * a specific method for each class.
    *
    * @param  other second antibody used in the function.
    * @return       <TT>true</TT> if the two antibodies are equal and
    *               <TT>false</TT> otherwise.
    */
  public boolean equals(Object other);

  /**
    * This function returns a string representation of the antibody.
    * To be accurate this function is not entirely needed, since it is
    * not used by the algorithm and since the class Object provides
    * a default implementation, but is is very useful to be able to
    * correctly visualize the results of the algorithm.
    *
    * @return       a string representation of the object.
    */
  public String toString();

  /**
    * This function computes the affinity between paratope of (this)
    * antibody and and the epitope the (other) antibody. For this
    * function to be effective it must take into account not only the
    * affinity between the two antibodies but also enhance this value
    * according to the relative quality of the recognizer paratope as
    * a possible solution to the problem.
    *
    * @param  other second antibody used in the function.
    * @return       a floating point value of the affinity.
    */
  public double Affinity(IAntibody other);

  /**
    * This function provides the basic recruitment capability for the
    * basic version of the Immunological Algorithm. Generates a mutated
    * copy of (this) antibody. Note that (this) antibody must not be
    * changed in the process.
```

```
    *
    * @return        the newly generated antibody.
    */
  public IAntibody Mutate();
}
```

A.2 IProblem interface

```
import java.util.Vector;

/**
 * This interface defines the minimum set of methods required by the
 * Immunological Algorithm to encode (together with {@link IAntibody})
 * a problem. The various constants and parameters selected by the
 * methods of this interface allow to customize the behavior of the
 * algorithm. Note that this interface is completely implementation
 * independent.
 *
 * @author       Jose Pacheco
 * @version      %I%, %G%
 */
public interface IProblem
{
  /**
    * This function generates the initial population of antibodies for a
    * given problem. Any method of generation can be applied, including
    * random generation, only taking care not to allow repetitions on the
    * population.
    *
    * @return     a Vector with the newly created population of antibodies.
    */
  public Vector InitialPopulation();

  /**
    * This function indicates whether the termination criterion for the
    * running of the algorithm has been reached. Based on the two
    * information factors provided (generation number and an array of
    * the concentrations) various criteria can be defined.
    *
    * @param  generation      current generation number.
    * @param  concentrations array of concentrations for the current
    *                        population.
```

```
    * @return                <TT>true</TT> if the termination criterion is
    *                         reached and <TT>false</TT> otherwise.
    */
  public boolean TerminationCondition(int generation, double[] concentrations);

  /**
    * This function returns the value of the C parameter. The parameter
    * C represents a rate constant that depends on the number of collisions
    * per unit of time and the rate of antibody production stimulated by a
    * collision.
    *
    * @return     the floating point value of this parameter.
    */
  public double GetC();

  /**
    * This function returns the value of the K1 constant. The constant
    * K1 represents the possible inequality between stimulation and
    * suppression. It used by the concentrations update rule of the
    * algorithm.
    *
    * @return     the floating point value of this constant.
    */
  public double GetK1();

  /**
    * This function returns the value of the K2 constant. The constant
    * K2 represents the tendency of cells to die in the absence of any
    * interactions. It used by the concentrations update rule of the
    * algorithm.
    *
    * @return     the floating point value of this constant.
    */
  public double GetK2();

  /**
    * This function returns the death threshold for antibodies.
    * Establishes the minimum concentration under what a given
    * antibody type must be eliminated from the population.
    *
    * @return     the floating point value of the threshold.
    */
  public double GetDeathTreshold();
```

```
  /**
    * This function returns the recruitment threshold for antibodies.
    * Indicates the minimum concentration above what a given antibody
    * is sufficiently stimulated to initiate the recruitment process.
    *
    * @return     the floating point value of the threshold.
    */
  public double GetRecruitmentTreshold();

  /**
    * This function returns the initial concentration for any newly
    * created antibody. It is used both when the initial population
    * is generated and on the recuitment process.
    *
    * @return     the floating point value of the initial concentration.
    */
  public double GetInitialConcentration();
}
```

A.3 ImmunoAlgorithm class

```
import java.util.Random;
import java.util.Vector;

/**
 * This class provides the framework to the use of a basic version of the
 * Immunological Algorithm. Together with interfaces {@link IAntibody} and
 * {@link IProblem} allows a simple and quick implementation of a problem.
 * The source code tries to be as generic and extensible as possible to
 * enable an easy undestanding and customization.
 *
 * @author        Jose Pacheco
 * @version       %I%, %G%
 */
public class ImmunoAlgorithm
{
  protected Random rand;

  /**
    * Generation counter.
    */
  protected int generation;
```

```
/**
  * Representation of the problem to be solved.
  */
protected IProblem problem;

/**
  * Vector of IAntibody objects.
  */
protected Vector population;

/**
  * Vector of antibody concentrations (Doubles).
  */
protected Vector concentrations;

/**
  * Contructor of the Immunological Algorithm class. Creates the
  * initial population and the corresponding vector of initial
  * concentrations.
  *
  * @param  probl representation of the problem to be solved.
  */
public ImmunoAlgorithm(IProblem probl)
{
  rand = new Random();

  generation = 0;
  problem = probl;
  population = problem.InitialPopulation();
  concentrations = new Vector();
  for (int i = 0; i < population.size(); i++)
    concentrations.addElement(
      new Double(problem.GetInitialConcentration()));
  Normalize();
}

/**
  * This function selects the value of the concentration of an antibody.
  *
  * @param  index index of the antibody in the population.
  * @return       the floating point value of the concentration.
  */
```

```
protected double GetConcentration(int index)
{
  return ((Double)concentrations.elementAt(index)).doubleValue();
}

/**
  * This function selects an antibody from the population.
  *
  * @param  index index of the antibody in the population.
  * @return       the selected antibody.
  */
protected IAntibody GetAntibody(int index)
{
  return (IAntibody)population.elementAt(index);
}

/**
  * This function normalizes the values of the concentrations of the
  * population. If the concentrations are perceived as percentages of
  * the total concentration of the system then their sum must be 1.0.
  * This function recalculates the concentrations so that this rule
  * is followed.
  */
protected void Normalize()
{
  double factor, sum = 0.0;

  for (int i = 0; i < population.size(); i++)
    sum += GetConcentration(i);

  factor = 1.0/sum;
  for (int i = 0; i < population.size(); i++)
    concentrations.set(i, new Double(GetConcentration(i)/factor));
}

/**
  * This function updates the concentrations of the population antibodies
  * according to the algorithm update rule:
  * <PRE>                                          N
  *          x (t+1) = x (t) + x (t) [ c . SUM (n   . x (t)) ]
  *           i         i       i          j=1   ij    j
  * </PRE>
  */
```

```
protected void UpdateConcentrations()
{
  double sum, nij, mij, mji, k1, k2, c, xi;
  int N = population.size();

  c = problem.GetC();
  k1 = problem.GetK1();
  k2 = problem.GetK2();
  for (int i = 0; i < N; i++)
  {
    sum = 0.0;
    for (int j = 0; j < N; j++)
    {
      mji = GetAntibody(i).Affinity(GetAntibody(j));
      mij = GetAntibody(j).Affinity(GetAntibody(i));
      nij = mji-k1*mij-k2/(c*N);

      sum += nij*GetConcentration(j);
    }
    xi = GetConcentration(i);
    xi += xi*c*sum;
    concentrations.set(i, new Double(xi));
  }
}

/**
  * This function removes from the population all the antibodies with
  * concentration inferior to the death threshold.
  */
protected void EliminateAntibodies()
{
  for (int i = 0; i < population.size(); )
    if (GetConcentration(i) < problem.GetDeathTreshold())
    {
      population.remove(i);
      concentrations.remove(i);
    }
    else
      i++;
}
```

```
/**
  * This function implements a simple recruitment strategy. The selected
  * strategy consists in probabilistic selecting antibodies with
  * concentrations superior to the recruitment threshold. The selected
  * antibodies are conned and a mutation of the clone is added (if new)
  * to the population.
  */
protected void Recruitment()
{
  int N= population.size();
  for (int i = 0; i <N; i++)
  {
    double xi = GetConcentration(i);
    if (xi > problem.GetRecruitmentTreshold() && rand.nextDouble() < xi)
    {
      IAntibody newAnti = GetAntibody(i).Mutate();
      if (!population.contains(newAnti))
      {
        population.addElement(newAnti);
        concentrations.addElement(
          new Double(problem.GetInitialConcentration()));
      }
    }
  }
}

/**
  * This function computes one step (one generation) of the
  * Immunological Algorithm:
  * <UL>
  *   <LI> Update concentrations using equation formula
  *   <LI> Eliminate antibodies under death threshold
  *   <LI> Recruitment of new antibodies
  * </UL>
  */
public void Step()
{
  generation++;
  UpdateConcentrations();
  Normalize();
  EliminateAntibodies();
  Recruitment();
}
```

```
/**
  * This function computes n steps of the algorithm.
  */
public void Step(int n)
{
  for (int i = 0; i < n; i++)Step();
}

/**
  * This function runs the Immunological Algorithm until the termination
  * condition is reached.
  */
public void Run()
{
  do
    Step();
  while (!problem.TerminationCondition(generation, GetConcentrations()));
}

/**
  * This function restarts the running process of the algorithm.
  * Creates a new initial population and the corresponding vector of
  * initial concentrations.
  */
public void ReStart()
{
  generation = 0;
  population = problem.InitialPopulation();
  concentrations = new Vector();
  for (int i = 0; i < population.size(); i++)
    concentrations.addElement(
      new Double(problem.GetInitialConcentration()));
  Normalize();
}

/**
  * This function selects the antibody with the highest concentration
  * in the population.
  *
  * @return        the selected antibody.
  */
```

```
  public IAntibody GetBest()
  {
    int bestIndex = 0;
    double bestConcentration = GetConcentration(0);

    for (int i = 1; i < population.size(); i++)
      if (GetConcentration(i) > bestConcentration)
        bestConcentration = GetConcentration(bestIndex = i);

    return GetAntibody(bestIndex);
  }

  /**
    * This function selects the concentrations of all the antibodies in
    * the population.
    *
    * @return        an array of doubles with the concentrations
    */
  public double[] GetConcentrations()
  {
    double[] cons = new double[concentrations.size()];

    for (int i = 0; i < concentrations.size(); i++)
      cons[i] = GetConcentration(i);

    return cons;
  }
}
```

Bibliography

[Ada and Nossal, 1987] Ada, G. L. and Nossal, S. G. (1987). The clonal-selection theory. *Scientific American*, 257(2):50–57.

[Axelrod, 1997] Axelrod, R. (1997). *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton Press.

[Balcazar et al., 1995] Balcazar, J., Díaz, J., and Gabarró, J. (1995). *Structural Complexity I*. Springer-Verlag New York, Inc., second edition edition.

[Bersini, 1992] Bersini, H. (1992). Reinforcement learning and recruitment mechanism for adaptive distributed control. Technical Report IR/IRIDIA/92-4, Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle.

[Bersini, 1994] Bersini, H. (1994). Reinforcement learning for homeostatic endogenous variables. Technical Report TR/IRIDIA/94-9, Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle.

[Bersini and Seront, 1993] Bersini, H. and Seront, G. (1993). Optimizing with the immune recruitment mechanism. Technical Report TR/IRIDIA/93-6, Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle.

[Bersini and Varela, 1991] Bersini, H. and Varela, F. J. (1991). The immune recruitment mechanism: A selective evolutionary strategy. Technical Report IR/IRIDIA/91-5, Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle.

[Burnet, 1959] Burnet, F. M. (1959). *The Clonal Selection Theory of Acquired Immunity*. Cambridge University Press.

[Burnet, 1960] Burnet, F. M. (1960). Immunological recognition of self. *Nobel Lecture*, pages 6–12.

[Celada and Seiden, 1992] Celada, F. and Seiden, P. E. (1992). A computer model of cellular interactions in the immune system. *Immunology Today*, 13(2):56–62.

[Coutinho, 1995] Coutinho, A. (1995). The network theory: 21 years later. *Scandinavian Journal of Immunology*, 42:3–8.

[DasGupta, 1998] DasGupta, D., editor (1998). *Artficial Immune Systems and Their Applications*. Springer-Verlag New York, Inc.

[de Castro and Timmis, 2002] de Castro, L. and Timmis, J. (2002). *Artificial Immune Systems: A New Computational Approach*. Springer-Verlag New York, Inc.

[D'haeseleer, 1996] D'haeseleer, P. (1996). An immunological approach to change detection: Theoretical results. In *The 9th IEEE Computer Security Foundations Workshop*.

[D'haeseleer et al., 1996] D'haeseleer, P., Forrest, S., and Helman, P. (1996). An immunological approach to change detection: Algorithms, analysis and implications. In *IEEE Symposium on Research in Security and Privacy*.

[Edelman, 1988] Edelman, G. M. (1988). *Neural Darwinism: The Theory of Neuronal Group Selection*. Basic Books, New York.

[Farmer et al., 1986] Farmer, J. D., Packard, N. H., and Perelson, A. S. (1986). The immune system, adaptation, and machine learning. *Physica*, 22D:187–204.

[Flanagan, 1999] Flanagan, D. (1999). *Java in a Nutshell (3rd ed.): A Desktop Quick Reference*. O'Reilly & Associates, Inc.

[Forrest et al., 1996] Forrest, S., Hofmaeyr, S. A., and Somayaji, A. (1996). Computer immunology. *Communications of the ACM*.

[Forrest and Longstaff, 1996] Forrest, S. and Longstaff, T. A. (1996). A sense of self for unix processes. In *IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press.

[Forrest and Perelson, 1992] Forrest, S. and Perelson, A. S. (1992). Computation and the immune system. *SIGBIO Newsletter, Association for Computing Machinery*, 12(2):505–530.

[Forrest et al., 1994] Forrest, S., Perelson, A. S., Allen, L., and Cherukuri, R. (1994). Self-nonself discrimination in a computer. In *IEEE Symposium on Research in Security and Privacy*.

[Forrest et al., 1997] Forrest, S., Somayaji, A., and Ackley, D. H. (1997). Building diverse computer systems. In *The 6th Workshop on Hot Topics In Operating Systems*, pages 67–72. IEEE Computer Society Press.

[Forsdyke, 1995] Forsdyke, D. R. (1995). The origins of the clonal selection theory of immunity as a case study for evaluation in science. *The FASEB Journal*, 9:164–166.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc.

[Herbert et al., 1995] Herbert, W. J., Wilkinson, P. C., and Stott, D. I. (1995). *The Dictionary of Immunology*. Academic Press, 4th edition.

[Hightower et al., 1995] Hightower, R., Forrest, S., and Perelson, A. S. (1995). The evolution of emergent organization in immune system gene libraries. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 344–350, San Francisco, CA. Morgan Kaufmann.

[Holland, 1975] Holland, J. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and, Artificial Intelligence*. MIT Press, 4th edition.

[Ishida, 1997] Ishida, Y. (1997). The immune system as a prototype of autonomous decentralized systems: An overview. In *3rd International Symposium on Autonomous Decentralized Systems (ISADS '97)*, pages 85–92.

[Jerne, 1955] Jerne, N. K. (1955). The natural-selection theory of antibody formation. *Proceedings of The National Academy of Sciences USA*, 41:849–856.

[Jerne, 1974] Jerne, N. K. (1974). Towards a network theory of the immune system. *Ann. Immunol. (Inst. Pasteur)*, 125(C):373–389.

[Kuby, 1997] Kuby, J. (1997). *Immunology*. W. H. Freeman Company, third edition edition.

[Lord, 2002] Lord, C. (2002). An emergent homeostatic model of immunological memory. Technical report, Carnegie Mellon Information Networking Institute.

[Martins, 2000] Martins, F. (2000). Computation with the immune system. Master's thesis, University of Azores.

[Maçãs, 1997] Maçãs, J. (1997). Computação no limiar do eu: Um novo algoritmo evolutivo. Master's thesis, Universidade de Lisboa.

[Paton, 1994] Paton, R., editor (1994). *Computing with Biological Metaphors: A Multidisciplinary Approach*. Chapman & Hall.

[Paul, 1993] Paul, W. E., editor (1993). *Fundamental Immunology*. Raven Press, New York, 3rd edition.

[Perelson and Weisbuch, 1997] Perelson, A. S. and Weisbuch, G. (1997). Immunology for physicists. *Reviews of Modern Physics*, 69(4):1219–1263.

[Poundstone, 1993] Poundstone, W. (1993). *Prisoner's Dilemma: John von Neumann, Game Theory and the Puzzle of the Bomb*. Doubleday Publishing.

[Rawlins, 1991] Rawlins, G. J. (1991). *Foundations of Genetic Algorithms*. Morgan Kaufmann.

[Reeves and Todd, 1996] Reeves, G. and Todd, I. (1996). *Lecture Notes on Immunology*. Blackwell Science, third edition edition.

[Rojas, 1996] Rojas, R. (1996). *Neural Networks: A Systematic Introduction*. Springer-Verlag New York, Inc.

[Siegelmann, 1999] Siegelmann, H. T. (1999). *Neural Networks and Analog Computation: Beyond the Turing Limit*. Birkhäuser.

[Somayaji et al., 1998] Somayaji, A., Hofmeyr, S., and Forrest, S. (1998). Principles of a computer immune system. In *Proceedings of the 1997 workshop on New security paradigms*, pages 75–82.

[Tarakanov et al., 2003] Tarakanov, A., Skormin, V., and Sokolova, S. (2003). *Immunocomputing: Principles and Applications*. Springer-Verlag New York, Inc.

[Varela and Coutinho, 1991] Varela, F. J. and Coutinho, A. (1991). Second generation immune networks. *Immunology Today*, 12(5):159–166.