Proc. of the ICONIP Conference (International Conference on Neural Information Processing), Workshop on Artificial Immune Systems, vol. **3**, pp. 1464-1468, Singapura 18-22 November, 2002.

IMMUNE, SWARM, AND EVOLUTIONARY ALGORITHMS PART I: BASIC MODELS

Leandro Nunes de Castro {*lnunes@dca.fee.unicamp.br*}
http://www.dca.fee.unicamp.br/~lnunes
Computer and Electrical Engineering School (FEEC)
State University of Campinas (Unicamp), Brazil

ABSTRACT

These two papers have three main aims. First (Part I), to review the general algorithms of immune, swarm and evolutionary systems. Second (Part II), to present a philosophical discussion about the similarities and differences between these paradigms, in terms of components, architecture, adaptation, interactions, and metaphors. Finally (Part II), to highlight the main features embodied in each approach, such that avenues for the creation of hybrid models can be suggested.

1. INTRODUCTION

All the algorithms to be discussed have in common a biological motivation and some application domains. This paper is divided into two parts – Part I and Part II – that describe the basic algorithms of each paradigm, and depict similarities and differences among them.

The algorithms are presented in a descriptive instead of mathematical form. No effort is made in the direction of presenting a large bibliographical survey, but (some of) the main textbooks of each field will be referenced as appropriate.

The focus of this part of the paper is on the immune paradigm, which, to date, still lacks a more thorough presentation.

2. EVOLUTIONARY ALGORITHMS

Evolution can be viewed as a change in the genetic composition of a population of individuals over time. In a simplified form, evolution is a result of the successive processes of reproduction and genetic variation followed by natural selection, which allows the fittest individuals to survive and reproduce, thus propagating their genetic material to future generations.

By mimicking the process of natural evolution, researchers developed the *evolutionary algorithms* (EA), which are based on the collective adaptability within a population of individuals, each of which represents a search point in the space of potential solutions to a given problem. In order for an evolutionary algorithm to work, a population of candidate solutions is initialized, and it evolves towards increasingly better regions of the search space by means of selection, reproduction and genetic variation mechanisms. The environment in which the population evolves is defined by the aim of the search, and delivers an information, termed *fitness*, that quantifies how good is an individual. The selection process favors the reproduction of individuals of higher fitness, and a recombination mechanism allows the mixing of parental information while passing it to their descendants. Finally, mutation introduces novelties in the population.

There are three main types of evolutionary algorithms [1], [2]: 1) evolution strategies; 2) genetic algorithms, and 3) evolutionary programming. Note that some authors consider genetic programming and classifier systems as other branches of the evolutionary algorithms; a discussion that will not be pursued here.

From a practical perspective, evolutionary algorithms are aimed at performing a search to identify an approximation of an (global) optimum of an objective function. The search is performed by evolving a population of individuals represented, in most cases, according to the application domain and type of evolutionary algorithm. The individuals of the population correspond to *chromosomes* that during the evolutionary process are allowed to suffer genetic variation and/or exchange genetic material. Most EAs follow the same standard sequence of steps, as described in Algo 1.

```
1. Initialization: randomly initialize a population of
   individuals.
2. Generation loop: apply the following evolutionary
   procedures of adaptation,
   2.1 Selection: individuals are selected for survival and
       reproduction according to their fitness.
   2.2 Reproduction and genetic variation: new
       individuals are created by recombining and/or
       introducing genetic variation into the selected
       individuals.
   2.3 Fitness evaluation: evaluate the fitness of each new
       individual in the population.
3. Cycle: repeat Step 2 until a given convergence criterion
   is met.
```

Algo 1: Standard evolutionary algorithm.

After Step 2 has been completed, a *generation* is said to have occurred, i.e., individuals reproduced giving rise to a new generation of individuals (offspring), and the process cycles.

3. SWARM ALGORITHMS

The expression "swarm intelligence" was coined in the late 1980's to refer to cellular robotic systems in which a collection of simple agents in an environment interact according to local rules. Several other authors have proposed and used similar definitions for swarm intelligence, such as:

> "Swarm intelligence is a property of systems of unintelligent agents of limited individual capabilities exhibiting collectively intelligent behaviors". [8]
>
> "[Swarm intelligence is] any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies." ([3]; p. 7).

In [7], the authors use the concept of a swarm in an even less restrictive form as a general expression that refers to any loosely structured collection of interacting elements.

Two main types of swarm intelligence algorithms can be found in the literature, namely *ant colony optimization* (ACO) [3], and *particle swarm optimization* (PSO) [7]. In the ACO paradigm, the authors showed how the very simple processes of foraging for food in ants could be used to solve combinatorial optimization problems. In contrast, the PSO algorithm simulates the ability of human societies to process knowledge. PSO algorithms also demonstrated to be powerful tools to solve optimization problems.

3.1. Particle Swarm Optimization

The particle swarm optimization algorithm was introduced to study social and cognitive behavior, but it has been largely applied as a problem-solving technique in engineering and computer science. There are two main versions of the PSO algorithm: a binary and a real-valued version. With the exception of the representation, the two versions of the algorithm are very much the same, thus only the real-valued (most popular) version will be discussed here.

The particle swarm approach assumes a population of individuals represented as binary strings or real-valued vectors – *particles*, which suffer an iterative procedure of adaptation to their environment. It also assumes that these individuals are social, what implies that they are capable of interacting with other individuals within a given neighborhood. The description given in this section follows that of [7].

There are two main types of information available to each individual of the population. The first is their own past experiences, and the second is the knowledge about how individuals around them have performed. The authors likened these two types of information to the individual learning and cultural transmission.

Individuals tend to be influenced by its success along its past history and also by the success of any individual in its neighborhood, i.e., with which it interacts. To these 'schemes of interactions' between individuals, the authors termed *sociometric principles*. Individuals can interact with each other in a number of ways. The simplest form is a binary interaction, where the individual interacts with its two nearest neighbors. Any number k of nearest neighbors can be used. If the number of nearest neighbors is less than the total number of individuals in the population, then this sociometric principle is called *lbest*, else (if $k = N$) it is called *gbest*. Conceptually, *gbest* connects all the individuals together, what means that its social interaction is maximal. In contrast, *lbest* results in a local neighborhood for the individual.

The authors claim that the binary particle swarm algorithm can be interpreted as a qualitative or quantitative social optimization algorithm, while the real-valued (continuous) version of PSO is a truly numeric optimization algorithm. In the latter version, the PSO searches for optima in R^n, where n is the dimension of the search space.

The continuous version of PSO assumes individuals as points in a space, and the change over time is represented as movements of the points, now defined as *particles*. A psychological system is viewed as an information processing function, and each coordinate of a particle in the search space corresponds to a psychological measure. Forgetting and learning are viewed respectively, as an increase or a decrease in the value of a given coordinate.

Assume that the position of a particle i is given by $\mathbf{x}_i$ and its velocity by $\mathbf{v}_i$. The velocity is a vector of numbers that are added to the position coordinates of the particle in order to move it throughout the search space along the iterations (t is the time index):

$$\mathbf{x}_i(t) = \mathbf{x}_i(t-1) + \mathbf{v}_i(t) \tag{1}$$

The social-psychological theory used as inspiration to develop the PSO algorithm suggests that individuals moving along a sociocognitive space should be influenced by their own previous behavior and by the successes of its neighbors. It is important to note that neighborhood is related to the topologic space that defines the sociometric structure of the population, not to the distance between individuals in the parameter space. In both versions (binary and continuous) of the algorithm, a neighborhood is defined for each individual based on its position in the topological population array. The population array is usually implemented as a ring structure, with the last member being a neighbor of the first one.

As the particles are moving in the space, the direction of movement is a function of its current position and velocity, the location of the individual's current best success $\mathbf{p}_i$, and the best position found by any member of the neighborhood $\mathbf{p}_g$:

$$\mathbf{x}_i(t) = f(\mathbf{x}_i(t-1), \mathbf{v}_i(t-1), \mathbf{p}_i, \mathbf{p}_g) \quad (2)$$

The change $\mathbf{v}_i$ in the trajectory of a particle is a function of the difference between the individual's previous best and current positions, and the difference between the neighborhood's best and the current individual's position. The formula for changing the velocity assumes continuous variables:

$$\mathbf{v}_i(t) = \mathbf{v}_i(t-1) + \varphi_1(\mathbf{p}_i - \mathbf{x}_i(t-1)) + \varphi_2(\mathbf{p}_g - \mathbf{x}_i(t-1)) \quad (3)$$

In order to avoid that this system explodes when the particles' oscillations become too large, the velocity of the particles is damped by a factor $V_{\max}$:

$$\text{if } v_{id} > V_{\max}, \text{ then } v_{id} = V_{\max}. \quad (4)$$

$$\text{if } v_{id} < -V_{\max}, \text{ then } v_{id} = -V_{\max}. \quad (5)$$

The general PSO algorithm is summarized in Algo 2.

```
1.  Initialization: randomly initialize a population of
    particles.
2.  Population loop: for each particle, do:
    2.1  Goodness evaluation and update: evaluate the
         'goodness' of the particle. If its goodness is
         greater than its best goodness so far, then this
         particle becomes the best particle found so far.
    2.2  Neighborhood evaluation: if the goodness of this
         particle is the best among all its neighbors, then
         this particle becomes the best particle of the
         whole neighborhood.
    2.3  Determine $\mathbf{v}_i$: apply equation (3).
    2.4  Particle update: apply the updating rule given by
         equation (1).
3.  Cycle: repeat Step 2 until a given convergence criterion
    is met.
```

Algo 2: Standard PSO algorithm.

3.2. Ant Colony Optimization

The other branch of swarm intelligence to be studied in this paper is composed of the so-called *ant algorithms*. Ant algorithms are multi-agent systems in which the behavior of each single agent, called *artificial ant* or simply *ant*, is inspired by the behavior of real ants [5]. The focus here will be on the *ant colony optimization* (ACO) algorithm, which by itself is a sub-branch of ant systems.

ACO algorithms take into account the fact that many ant species have trail-laying trail-following behavior when foraging. Individual ants deposit a chemical substance called *pheromone* as they move from a food source to their nest, and foragers follow the pheromone trail. An important aspect of the foraging strategies of ant colonies is that the collective action of many ants results in the location of the shortest path between a food source and a nest [3]. The basic idea underlying the ACO algorithms is that good 'solutions' (usually corresponding to paths from the nest to the food source) are reinforced by a virtual pheromone trail laid by individual ants. In addition, pheromones evaporate allowing the exploration of other food sources.

ACO algorithms are usually applied to discrete (combinatorial) optimization problems. Assuming that the problem to be optimized can be represented by a graph, the general ACO algorithm can be described as in Algo 3.

```
1.  Initialization: assign the same initial pheromone value
    to each edge of the graph, and randomly place an ant in
    a location of the search space.
2.  Population loop: for each ant, do:
    2.1  Probabilistic transition rule: according to a given
         probabilistic transition rule, move an ant over the
         space so that a solution to the problem is built.
    2.2  Goodness evaluation: evaluate the goodness of
         the solution obtained by this ant.
    2.3  Pheromone updating: update the pheromone
         level of each edge by reinforcing good solutions.
         Reduce the pheromone level of each edge
         (evaporation).
3.  Cycle: repeat Step 2 until a given convergence criterion
    is met.
```

Algo 3: Standard ACO algorithm.

4. ARTIFICIAL IMMUNE SYSTEMS

Artificial immune systems (AIS) have been defined as adaptive systems inspired by the immune system and applied to problem solving [4].

In a simplified form, to design an AIS it is necessary to choose an appropriate *shape-space* for the components of the system, one or more *affinity measure*(*s*), and an *immune algorithm*.

The shape-space is a formalism used to create abstract ('artificial') representations for the components of the immune system. The 'shape' of an immune cell or molecule corresponds to all the features required to quantify interactions between the cell or molecule and the environment, and also with other elements of the system. There are four main types of shape-spaces: Euclidean or real-valued, Hamming, Integer, and Symbolic. In Euclidean shape-spaces the elements of the system are represented as real-valued vectors. In Hamming shape-spaces the elements of the system are attribute strings built out of a finite alphabet. In Integer shape-spaces, cells and molecules are represented as integer values. (Note that Integer shape-spaces are a particular case of

Hamming shape-spaces.) Symbolic shape-spaces use different types of attributes to represent a single element, for example, an integer value and a string such as 'color'.

There are two main types of interactions that can be performed by an element of an artificial immune system. One is the interaction with the environment. For example, an AIS can be used as a pattern recognition tool, thus the 'artificial immune cells' are used to recognize a set (or sets) of 'artificial antigens' (patterns). In this case, the degree of recognition, known as the *affinity* between the immune cell and the antigen, is measured via a function that quantifies the strength of the match between the two. If we assume that two cells interact to the extent their 'shapes' are similar, then a similarity measure can be used according to the shape-space adopted. As an example, assume an immune cell with the following shape $Ab = [1,0,0,0,1]$ in a binary Hamming shape-space, and an antigen with the following shape $Ag = [0,1,1,1,1]$. If affinity is directly proportional to their similarity, then the expression L – *Hamming distance* is a suitable measure to quantify immune recognition, where L is the length of the string. Their affinity in this case is $Aff = 5 - 4 = 1$. There are several types of affinity measures, which usually vary according to the shape-space adopted, which by itself is usually defined by the problem in hand.

The last 'building block' of artificial immune systems corresponds to the immune algorithms. There are a number of different algorithms that can be applied to many domains, from data analysis to autonomous navigation. These immune algorithms were inspired by works on theoretical immunology and several processes that occur within the immune system. They can be classified as *population-based* and *network-based* immune algorithms. In population-based algorithms, the elements of the system are not connected with each other, meaning that they only interact directly with the environment. Interactions with other elements of the system can only be performed indirectly, via, for example, a reproductive operator. In network-based AIS by contrast, some (or all) elements of the system are interconnected. This way, there are two levels of interaction within this system: with the environment and with other elements in the system.

4.1. Immune Algorithms

In order not to overload the text with descriptions of several immune algorithms, the focus will be given to two population-based AIS (negative and clonal selection algorithms), and to two types of network-based AIS (continuous and discrete immune networks).

The main role played by the immune system is to protect our organisms against infectious diseases (caused by viruses, bacteria, etc.), and to eliminate debris and malfunctioning cells. To perform these functions, the immune system has to be able to distinguish between our own cells (known as *self*) and those elements that do not belong to the organism itself (known as *nonself*). One of the processes by which the immune system differentiates self from nonself (*self/nonself discrimination*) is termed *negative selection*. This gave rise to the *negative selection algorithm*. After distinguishing between self and nonself, the immune system has to perform an immune response in order to eliminate the nonself substances. *Clonal selection* is the name given to a theory that explains how the immune cells and molecules cope with invading nonself elements, known as *antigens*.

Assuming that the set of self elements is known, the standard negative selection algorithm aims at generating another set of immune cells, known as *detectors*, that recognizes any cell (pattern) but those belonging to the self set. The algorithm is summarized in Algo 4.

```
1. Initialization: randomly generate a number of candidate
   detectors (attribute strings).
2. Censoring: while a set of detectors of a given size has
   not yet been produced, do:
   2.1 Affinity evaluation: determine the affinity
       between every self and a candidate detector.
   2.2 Selection: if the candidate detector recognizes
       any element of the self this candidate is
       eliminated. Else, place this candidate detector in
       the detector set.
3. Monitoring: after the set of detectors has been
   generated, monitor a new set of self for any variation. It
   means that if any element of the detector set matches
   an element of the new self-set, then a nonself element
   was detected.
```

Algo 4: Standard negative selection algorithm.

The theory known as clonal selection is used to explain how the immune system 'fights' against an antigen. When a bacterium invades our organism, it starts multiplying and damaging our cells. One form the immune system found to cope with this replicating antigen was by replicating the immune cells successful in recognizing and fighting against this disease-causing element. Those cells capable of recognizing the antigen reproduce themselves asexually in a way proportional to their degree of recognition: the better the antigenic recognition, the higher the number of offspring (clones) generated. During the process of cell division (reproduction), individual cells suffer a mutation that allows them to become more adapted to (increase affinity with) the antigen recognized: the higher the affinity of the parent cell, the lower the mutation they suffer. Assuming in this case a set of antigens to be recognized, a basic clonal selection algorithm, named CLONALG, works as in Algo 5.

The other class of immune algorithms is composed of the so-called immune networks. In the year 1974, N. Jerne [6] hypothesized that the immune cells and molecules are capable of recognizing each other in addition to recognizing invading antigens. This idea had a great impact on immunology at that time because as a result of this internal recognition process the immune system would present a dynamic behavior even if it is not being stimulated by an external invader (antigen).

```
1. Initialization: randomly initialize a repertoire
   (population) of attribute strings (immune cells).
2. Population loop: for each antigen, do:
   2.1 Selection: select those cells whose affinities with
       the antigen are greater.
   2.2 Reproduction and genetic variation: generate
       copies of the immune cells: the better each cell
       recognizes the antigen, the more copies are
       produced. Mutate (perform variations) in each
       cell inversely proportional to their affinity: the
       higher the affinity, the smaller the mutation rate.
   2.3 Affinity evaluation: evaluate the affinity of each
       mutated cell with the antigen.
3. Cycle: repeat Step 2 until a given convergence criterion
   is met.
```

Algo 5: Standard clonal selection algorithm.

According to the immune network theory, immune cells have portions of their receptor molecules that can be recognized by other immune cells in a way similar to the recognition of an invading antigen. This results in a network of communication (recognition) between immune cells. When an immune cell recognizes an antigen or another immune cell, it is stimulated. On the other hand, when an immune cell is recognized by another immune cell, it is suppressed. The sum of the stimulation and suppression received by the network cells, plus the stimulation by the recognition of an antigen corresponds to the stimulation level S of a cell, as described by Equation 6:

$$S = N_{st} - N_{sup} + A_s, \qquad (6)$$

where N_{st} corresponds to the network stimulation, N_{sup} is the network suppression, and A_s corresponds to the antigenic stimulation.

```
1. Initialization: initialize a network of immune cells.
2. Population loop: for each antigen, do:
   2.1 Antigenic recognition: match network cells
       against the antigen.
   2.2 Network interactions: match network cells
       against network cells.
   2.3 Metadynamics: introduce new cells into the
       network and eliminate useless ones (based on a
       certain criterion).
   2.4 Stimulation level: evaluate the stimulation level
       of each network cell taking into account the
       results of the previous steps (Equation (6)).
   2.5 Network dynamics: update the network's
       structure and free parameters according to the
       stimulation level of individual cells.
3. Cycle: repeat Step 2 until a given convergence
   criterion is met.
```

Algo 6: General immune network algorithm.

The stimulation level of an immune cell is going to determine its probability of reproduction and genetic variation. There are two main types of immune network models: continuous and discrete. Continuous immune networks have their repertoires of cells governed by one or more sets of ordinary differential equations (ODE), each of which corresponds to the variation in number and affinity of a given immune cell. Discrete immune networks are governed by a difference equation that is used in an iterative procedure of adaptation controlling the number and affinities of individual cells. A general immune network algorithm can be proposed as in Algo 6.

5. SUMMARY

This first part of the paper provided a descriptive review of evolutionary, swarm and immune approaches, focusing on the main algorithms and ideas of each paradigm. The algorithms were standardized (portions of the algorithms are usually modified, skipped, and added in particular studies) so as to make it possible a general discussion about the similarities and differences of the approaches, to be presented in the next part (Part II) of this work.

ACKNOWLEDGMENTS

The author thanks CNPq (Profix, Proc. n. 540396/01-0) for the financial support.

6. REFERENCES

[1] Bäck, T., Fogel, D. B. & Michalewicz, Z. (2000a), *Evolutionary Computation 1 Basic Algorithms and Operators*, Institute of Physics Publishing (IOP), Bristol and Philadelphia.

[2] Bäck, T., Fogel, D. B. & Michalewicz, Z. (2000b), *Evolutionary Computation 2 Advanced Algorithms and Operators*, Institute of Physics Publishing (IOP), Bristol and Philadelphia.

[3] Bonabeau, E., Dorigo, M. & Theraulaz, G. (1999), *Swarm Intelligence From Natural to Artificial Systems*, Oxford University Press.

[4] de Castro, L. N. & Timmis, J. (2002), *Artificial Immune Systems: A New Computational Intelligence Approach*, Springer-Verlag, London.

[5] Dorigo, M. & Di Caro, G. (1999), "The Ant Colony Optimization Meta-Heuristic", In *New Ideas in Optimization*, D. Corne, M. Dorigo & F. Glover (eds.), McGraw-Hill, pp. 11-32.

[6] Jerne, N. K. (1974), "Towards a Network Theory of the Immune System", *Ann. Immunol.* (Inst. Pasteur) **125C**, pp. 373-389.

[7] Kennedy, J., Eberhart, R. & Shi, Y. (2001), *Swarm Intelligence*, Morgan Kaufmann.

[8] White T. & Pagurek B. (1998), "Towards Multi-Swarm Problem Solving in Networks", *Proc. of the Third International Conference on Multi-Agent Systems* (ICMAS '98), pp 333-340.