Application of Neumann's Machine of Self-Reproduction - Evolution and Artificial Life in Process Engineering

János Madár, János Abonyi, and Ferenc Szeifert

Department of Process Engineering, University of Veszprém,
Veszprém, H-8200, P.O. Box. 158., Hungary,
http://www.fmt.vein.hu/softcomp, abonyij@fmt.ven.hu

Abstract: The aim of this paper is to present the historical background of artificial life based stochastic optimization algorithms from the viewpoint of Neumann's Self-reproduction scheme. The theory of evolutionary and swarm optimization are overviewed and compared from this viewpoint. A detailed application example is given to demonstrate how these tools can be applied to solve process optimization problems. The performances of Evolutionary Strategy, Particle Swarm Optimization, Simulated Annealing and Sequential Quadratic Programming based algorithms are compared.

Keywords: Neumann's Self-reproduction scheme, Artificial life, Evolutionary strategies

1. Introduction

Artificial life (A-Life) attempts to understand the essential general properties of living systems by synthesizing life-like behaviour in software, hardware and other human-made systems. A-Life includes two-folded research topic: (1) A-Life studies how computational techniques can help when studying biological phenomena. (2.) A-Life studies how biological techniques can help out with computational problems. The second research topic has leaded to such algorithms as Evolutionary Algorithms and Particle Swarm optimization, which are the most popular stochastic optimization techniques.

If any person could be called the father of A-life, it would be John von Neumann. His well known self-reproduction scheme can be interpreted as the general version of the previously mentioned stochastic optimization techniques. The aim of this paper is to discuss the common backgrounds of these algorithms and demonstrate how these tools can be applied in process engineering. The presented situational examples and tools can be downloaded from the website of the author.

2. Neumann's Machine of Self-Reproduction

In the late 1940's John von Neumann began to work on what he intended as a comprehensive "theory of complex automata". He started to work on a book length manuscript on this subject in 1952. However, he put this aside in 1953, apparently due to pressure of other work. This manuscript was eventually edited, and combined for publication with some related lecture transcripts, by Bruks in 1966 [1,2]. In this work von Neumann developed his famous logical model of self-reproduction as an answer to his observation that, unlike machines, biological organisms have an ability to self-replicate while increasing their complexity without limit.

As opposed to this natural self-reproduction, mechanical artefacts are produced via more complicated factories and can only degenerate in their complexity. Neumann was searching for systems that may self-reproduce while possibly increasing their complexity. Neumann's model is based on a memory-stored description, *PHI(X),* that can be interpreted by a **universal constructor automaton** *A* to produce any automaton *X*; if a description of *A*, *PHI(A)*, is fed to *A* itself, then a new copy of *A* is obtained. In addition to the universal constructor, an automaton *B* capable of **copying** any description, *PHI(X)*, is included in the self-replication scheme. A third automaton *C* is also included to effect all the manipulation of descriptions necessary - a sort of **operative system**. To sum it up, the self-replicating system contains the set of automata *(A + B + C)* and a description *PHI(A + B + C).* The description, or program, is used in two different ways: it is both translated and copied. In the first role, it controls the construction of an automaton by causing a sequence of activities (active role of description). In the second role, it is simply copied (passive role of description). Perhaps the most important consequence of the requirement of memory-based descriptions in Von Neumann's self-reproduction scheme is its opening the possibility for open-ended emergent evolution. As Von Neumann discussed, if the description of the self-reproducing automata is **changed (mutated)**, in a way as to not affect the basic functioning of *(A + B + C)* – that is, if the semantic closure in not destroyed – then, the new automaton *(A + B + C)`* will be slightly different from its parent. Von Neumann used a new automaton *D* to be included in the self-replicating organism, whose function does not disturb the basic performance of *(A + B + C)*; if there is a mutation in the *D* part of the description, say *D`*, then the system *(A + B + C + D) + PHI(A + B + C + D`)* will produce *(A + B + C + D`) + PHI(A + B + C + D`)*. Von Neumann further proposed that non-trivial self-reproduction should include the "ability to undergo inheritable **mutations** as well as the ability to make another organism like the original", to distinguish it from "naive" self-reproduction.

Since computers were nothing more than information driven machines Neumann felt that a computer could eventually emulate life by passing its information along to a new generation of regenerating computers. Just years after von Neumann postulated this theory, Watson and Crick confirmed his beliefs when they discovered the structure and nature of DNA. It has turned out, the ability to transmit mutations is precisely at the core of the principle of natural selection of modern Darwinism. In principle, if the language of description is rich enough, an endless variety of organisms can be evolved.

Neumann was never more than a theorist in Artificial Life. He imagined an incredible creature that lived in an environment with infinite resources. The creature itself was very intricate, being made up of cells with 29 different states. At the time he envisioned this creature the technology was not available to create it. Even today the undertaking may not be possible.

However, these works initialized non-gradient based, probabilistic search algorithms that are generally mimic some natural phenomena, for example evolutionary algorithms and simulated annealing, and swarm intelligence. Evolutionary algorithms model the evolution of a species, based on Darwin's principle of survival, simulated annealing is based on statistical mechanics and models the equilibrium of large numbers of atoms during an annealing process, while swarm intelligence is the property of a system whereby the collective behaviours of (unsophisticated) agents interacting locally with their environment cause coherent functional global patterns to emerge.

In the remaining paper these algorithms will be overviewed and applied, and some discussion will be given how these tools are connected to the original self-replication scheme of John von Neumann.

3. Evolutionary Algorithms

The Evolutionary Algorithm (EA) is an optimization method which uses the previously presented computational model of natural selection. EAs work with a population of potential solutions to a problem, where each individual within the population represents a particular solution, generally represented in some form of **genetic code**. A single process engineering problem can contain a mixture of decision variable formats (numbers, symbols, and other structural parameters). Since the EA operates on a “genetic” encoding of the optimized variables, diverse types of variable can be simultaneously optimized. The **fitness value** of the individual expresses how good the solution is at solving the problem. Better solutions are assigned higher values of fitness than worse performing solutions. The key of EA is that the fitness also determines how successful the individual will be at propagating its genes (its code) to subsequent generations.

Table 1. A typical evolutionary algorithm

```
procedure EA;    {
        Initialize population;
        Evaluate all individuals;
        while (not terminate) do    {
                Select individuals;
                Create offspring from selected individuals
                        using Recombination and Mutation;
                Evaluate offspring;
                Replace some old individuals by some offspring;
        }
}
```

In practical system identification, process optimization or controller design it is often desirable to simultaneously handle several **objectives and constraints**. For the purposes of the EA, these must be combined to form a single fitness value. The weighted-sum approach has proved popular in the literature, since it is amenable to a solution by conventional EA methods, but Pareto-based multi-objective techniques are likely to surpass this in the future.

The **population** is evolved over generations to produce better solutions to the problem. The evolution is performed using a set of stochastic genetic operators, which manipulate the genetic code used to represent the potential solutions. Most evolutionary algorithms include operators that select individuals for reproduction, produce new individuals based on those selected, and determine the composition of the population at the subsequent generation.

Table 1. outlines a typical EA. A population of individuals is randomly initialized and then evolved from generation to generation by repeated applications of evaluation, selection, mutation and recombination.

In the selection step, the algorithm selects the parents of the next generation. The population is subjected to "environmental pressure", which means the selection of the fittest individuals. The most important automated selection methods are Stochastic Uniform Sampling, Tournament Selection, Fitness Ranking Selection and Fitness Proportional Selection.

After the selection of the individuals, the new individuals of the next generation (also called offspring) are created by recombination and mutation.

- The *recombination* (also called crossover) exchanges information between two selected individuals to create one or two new offspring.
- The *mutation* operator makes small, random changes to the genetic coding of the individual.

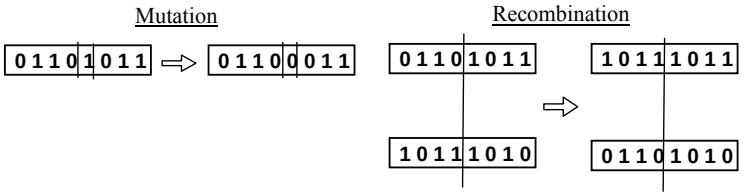

Figure 1. Mutation and recombination of binary strings

The final step of the evolutionary procedure is the replacement, when the new individuals are inserted into the new population. Once the new generation has been constructed, the processes that result in the subsequent generation of the population are begun once more.

Since, an EA search is directed and, represents potentially much greater efficiency than a totally random or enumerative search [3]. The main benefit of the EA is that it can be applied to a wide range of problems without significant modification. However, it should be noted that EA has several implementations: Evolutionary Programming (EP), Evolutionary Strategy (ES), Genetic Algorithm (GA) and Genetic Programming (GP), and the selection of the proper technique and the tuning of the parameters of the selected technique require some knowledge about these techniques.

The GA, as originally defined by John Holland and his students in the 1960s [4], uses bit-string representation of the individuals. Depending on the problem, the bit-strings (chromosomes) can represent numbers or symbols. The recombination means the swapping of strings fragments between two selected parents (see Figure 1), while the mutation means the flip of a few bits in these strings.

The recombination has much bigger probability than the mutation, so the recombination is often said to be "primary searching operator" [4]. GA applies simple replacement technique: all the original individuals are replaced by the created offspring; expect in case of the *elitist strategy* when some of the best individuals are also placed into the next generation.

Contrary to GA, ES typically searches in continuous space. The main difference from GA is that ES uses real-valued representation of the individuals. The individuals in ES are represented by n-dimensional vectors ($\mathbf{x} \in \Re^n$), often referred as object variables.

To allow for a better adaptation to the particular optimization problem, the object variables are accompanied by a set of the so-called strategy variables. Hence, an individual $\mathbf{a}_j = (\mathbf{x}_j, \boldsymbol{\sigma}_j)$ consists of two components, the object variables, $\mathbf{x}_j = \lfloor x_{j,1}, ..., x_{j,i} \rfloor$, and strategy variables, $\boldsymbol{\sigma}_j = \lfloor \sigma_{j,1}, ..., \sigma_{j,n} \rfloor$.

Because in the nature small changes occur frequently but large ones only rarely, as mutation operator normally distributed random numbers are added to the individuals:

$$x_{j,i} = x_{j,i} + N\left(0, \sigma_{j,i}\right) \tag{1}$$

Before the update of the object variables, the strategy variables are also mutated using a multiplicative normally distributed process:

$$\sigma_{j,i}^{(t)} = \sigma_{j,i}^{(t-1)} \exp\left(\tau' N\left(0,1\right) + \tau N_i\left(0,1\right)\right) \tag{2}$$

with $\exp\left(\tau' N\left(0,1\right)\right)$ as a global factor which allows an overall change of the mutability and $\exp\left(\tau N_i\left(0,1\right)\right)$ allowing for individual changes of the mean step sizes. The parameters can be interpreted in the sense of global learning rates. Schwefel suggests to set them as

$$\tau' = \frac{1}{\sqrt{2n}} \quad \tau = \frac{1}{\sqrt{2\sqrt{n}}} \tag{3}$$

Recombination in ES can be either sexual (local), where only two parents are involved in the creation of an offspring, or global, where up to the whole population contributes to a new offspring. Traditional recombination operators are discrete recombination, intermediate recombination, and geometric recombination, all existing in a sexual and global form. When F and M denote two randomly selected individuals from the μ parent population, the following operators can be defined [5,6]:

$$x_i' = \begin{cases} x_{F,i} & \text{no recombination} \\ x_{F,i} \text{ or } x_{M,i} & \text{discrete} \\ \left(x_{F,i} + x_{M,i}\right)/2 & \text{intermediate} \\ \sum_{k=1}^{\mu} x_{k,i} / \mu & \text{global avarge} \end{cases} \qquad \sigma_i' = \begin{cases} \sigma_{F,i} & \text{no recombination} \\ \sigma_{F,i} \text{ or } \sigma_{M,i} & \text{discrete} \\ \left(\sigma_{F,i} + \sigma_{M,i}\right)/2 & \text{intermediate} \\ \sum_{k=1}^{\mu} \sigma_{k,i} / \mu & \text{global avarge} \end{cases} \tag{4}$$

The selection is stochastic in the ES. First we chose the best μ individuals to be parents, and then we select the parent-pairs uniformly randomly from these individuals.

Somewhat different to the generic evolutionary algorithm, selection is performed after the genetic operators have been applied. The standard notations in this domain, (μ+λ)-ES and (μ,λ)-ES, denote algorithms in which a population of μ parents generates λ offspring. The next generation is created by selecting the fittest m individuals. In the case of (μ,λ)-ES only the λ offspring are considered for selection, thus limiting the 'life' of an individual to one generation, while in the (μ+λ)-ES the μ parents are also considered for selection.

Evolutionary Programming (EP) was developed by Fogel et al. [7] independently from ES. Originally, it was used to achieve machine intelligence by simulated evolution. In the EP there are μ individuals in every generation. Every individual is selected and mutated (there is no recombination). After the calculation of the fitness values of the new individuals, μ individuals are selected to form the next generation from the μ+μ individuals, using a probabilistic function based on the fitness of the individuals.

4. Particle Swarm Optimization

4.1. The Algorithm

There are two popular swarm inspired methods in computational intelligence areas: Ant colony optimization (ACO) and particle swarm optimization (PSO). ACO was inspired by the behaviours of ants and has many successful applications in discrete optimization problems. The particle swarm concept originated as a simulation of simplified social system. The original intent was to graphically simulate the choreography of bird of a bird block or fish school. However, it was found that particle swarm model can be used as an optimizer. Suppose the following scenario: a group of birds are randomly searching food in an area. There is only one piece of food in the area being searched. All the birds do not know where the food is. But they know how far the food is in each iteration. So what's the best strategy to find the food? The effective one is to follow the bird which is nearest to the food.

Particle swarm optimization (PSO) is based on this scheme. This stochastic optimization technique has been developed by Eberhart and Kennedy in 1995 [8]. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. All of particles have fitness values which are evaluated by the fitness function to be optimized, and have velocities which direct the flying of the particles.

PSO is initialized with a group of random particles (solutions) and then searches for optima by updating generations. In every iteration, each particle is updated by following two "best" values. The first one is the best solution (fitness) it has achieved so far. (The fitness value is also stored.) This value is called pbest. Another "best" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is a global best and called gbest.

When a particle takes part of the population as its topological neighbors, the best value is a local best and is called lbest.

$$\mathbf{v}_j(k+1) = w \cdot \mathbf{v}_j(k) + c_1 \cdot rand() \cdot \left(\mathbf{x}_{pbest} - \mathbf{x}_j(k)\right) + c_2 \cdot rand() \cdot \left(\mathbf{x}_{gbest} - \mathbf{x}_j(k)\right) \quad (5)$$

$$\mathbf{x}_j(k+1) = \mathbf{x}_j(k) + \mathbf{v}_j(k+1) \cdot dt \quad (6)$$

where *v* is the particle velocity, *persent* is the current particle (solution), *pbest* and *gbest* are defined as stated before, *rand()* is a random number between [0,1), c_1, c_2 are learning factors usually $c_1 = c_2 = 2$. Table 2.

shows the pseudo code of the PSO algorithm.

The role of the, *w*, inertia weight in Eq. (5), is considered critical for the PSO's convergence behaviour. The inertia weight is employed to control the impact of the previous history of velocities on the current one. Accordingly, the parameter regulates the trade–off between the global and local exploration abilities of the swarm. A large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration, i.e. fine–tuning the current search area.

4.2. PSO vs. ES

As can be seen, PSO shares many similarities with evolutionary computation techniques. Both algorithms start with a group of a randomly generated population, both have fitness values to evaluate the population. Both update the population and search for the optimum with random techniques. Both systems do not guarantee success.

The main difference between these algorithms is that PSO does not have genetic operators like crossover and mutation. Particles update themselves with the internal velocity. They also have memory, which is important to the algorithm.

Table 2.: PSO algorithm

```
procedure PSO; {
   Initialize particles;
   while (not terminate) do {
         for each particle {
            Calculate fitness value;
            if fitness < pBest than  pBest = fitness;
         }
         Choose the best particle as the gBest;
         for each particle {
            Calculate particle velocity;
            Update particle position;
         }
   }
}
```

Compared with evolutionary algorithms, the information sharing mechanism in PSO is significantly different. In EAs, chromosomes share information with each other. So the whole population moves like a one group towards an optimal area. In PSO, only gBest (or lBest) gives out the information to others. It is a one -way information sharing mechanism. The evolution only looks for the best solution. Compared with EAs, all the particles tend to converge to the best solution quickly even in the local version in most cases.

Compared to EA, the advantages of PSO are that PSO is easy to implement and there are few parameters to adjust. Hence, PSO has been successfully applied in many areas: function optimization, artificial neural network training, fuzzy system control, and other areas where GA can be applied. In the following section we will demonstrate how this tool can be applied in process optimizations.

5. Application Example

The performance of the proposed optimization techniques are illustrated in the model-based temperature profile optimization of beer fermentation. During the beer fermentation a temperature profile is applied to drive the process so as to obey to certain constraints. The design of this temperature profile is an optimization problem where the objective is to optimize the quality of the beer.

In this paper a kinetic model published by Andres-Toro [9] has been used to estimate the effect of the temperature profiles. This model has been developed from experimental data and shows good results in the aspect of a realistic view of the fermentation process. The model takes into account seven components: three components of the biomass (latent, active, dead), ethanol and sugar, and two important byproducts: ethyl acetate and diacetyl. The model equations and parameters are taken from the article of Carrillo-Ureta [10].

A good temperature profile should result in high ethanol, low sugar and ethyl acetate concentrations, a very low diacetyl and biomass concentrations, and relatively smooth temperature profile. Hence we used the next performance index:

$$J = C_{ethanol} - 2C_{acetat} - 10C_{diacetyl} - 10C_{biomass} \tag{7}$$

where the C_x denotes the final concentration of x component (the operation time is 150 hours). The optimization goal is maximization of (7) performance index.

In order to apply the optimization algorithms for this problem, it is necessary to design a suitable representation of the temperature profile. For this purpose a simple but effective method has been developed. The profile is divided to 20 segments, defined by 22 knots. Hence, 42 design variables of the optimization

problem are the time values and temperatures of breakpoints for the possible piecewise-linear trajectories.

To analyze the Artificial Life algorithms, three tests were performed: nonlinear Sequential Quadratic Programming (SQP), Evolutionary Strategy (ES) and Particle Swarm Optimization (PSO). The function evaluation number was limited to 2500, which was enough large for this problem.

Beside these tools the simulated annealing has been also applied to this problem. Simulated annealing is based on an analogy with thermodynamics, specifically with the way that liquids freeze and crystallize, or metals cool and anneal. This general randomization technique can help to avoid the problem of getting stuck in a local minimum and to lead towards the globally optimum solution. The applied adaptive simulated annealing (ASA) is based on Monte Carlo importance-sampling technique for doing large-dimensional path integrals arising in statistical physics problems (also influenced by Neumann) [13]. All of these algorithms have been implemented in MATLAB, and the code can be downloaded from the website of the author. Table 3. shows the results of the four methods.

Table 3.: Resulted performance indexes achieved by SQP, ASA, ES and SPO

	SQP	ASA	ES	**PSO**
Performance (J)	38.8536	39.12	39.1648	**39.1755**

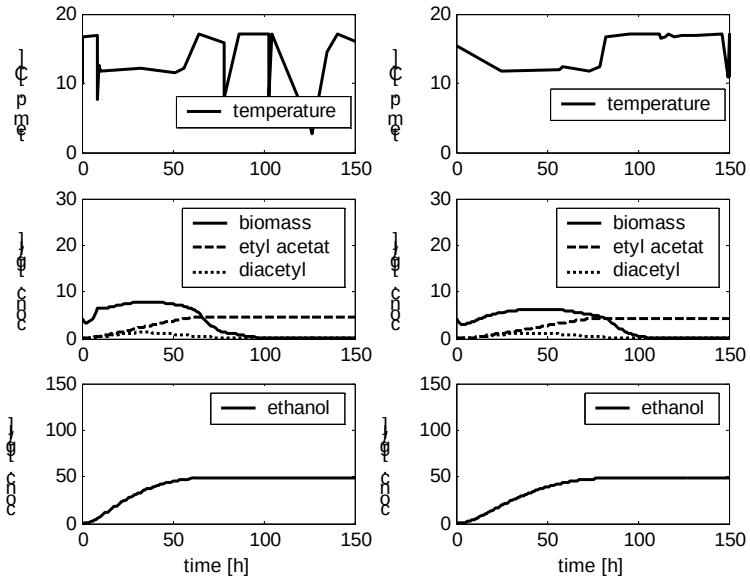

Figure 2.: Resulted temperature and concentration profiles
by SQP (left) and SPO (right)

6. Conclustions

Von Neumann believed in the logic behind life. In fact he viewed life as a complex machine, as many have done before him (such as Descartes) [11]. He wanted to create life (artificially) himself, or at least machines based on the concepts of life. He approached this problem by simplifying life, but he didn't carry this far enough, since his construct was (and still is!) too complex for any reasonable application. Hence, von Neumann was never more than a theorist in Artificial Life. He imagined an incredible creature that lived in an environment with infinite resources. The creature itself was very intricate, being made up of cells with 29 different states. At the time he envisioned this creature the technology was not available to create it. Even today the undertaking may not be possible. However, these works initialized non-gradient based, probabilistic search algorithms that are generally mimic some natural phenomena, for example genetic algorithms and simulated annealing.

In this paper, a fairly recent type of probabilistic search algorithms, called Particle Swarm Optimization (PSO) and Evolutionary algorithms were investigated. The underlying idea of these algorithms is the separation of solutions for a particular problem (e.g. a machine) from descriptions of those solutions (memory). These algorithms work on these descriptions and not on the solutions themselves, that is, variation is applied to descriptions, while the respective solutions are evaluated, and the whole (description-solution) selected according to this evaluation. Such machine/description separation follows aspects of von Neumann's self-reproducing scheme which is able to increase the complexity of the machines described. However, the form of organization attained by these algorithms is not self- organizing in the sense of a boolean network of cellular automata. Even though the solutions are obtained from the interaction of a population of elements, and in this sense following the general rules usually observed by computationally emergent systems, they do not self- organize since they rely on the selective pressures of some fitness function. The order so attained is not a result of the internal dynamics of a collection of interacting elements (like a random net), but is instead dictated by the external selection criteria. In this sense, ES and PSO follow a memory-based selective organization scheme.

Although the presented probabilistic search algorithms generally require many more function evaluations to and an optimum solution, as compared to gradient based algorithms, they do provide several advantages. These algorithms are generally easy to program, can efficiently make use of large numbers of processors, do not require continuity in the problem definition, and generally are better suited for finding a global, or near-global, solution. In particular these algorithms are ideally suited for solving discrete and/or combinatorial type optimization problems, and have proved particularly successful in problems that are difficult to formalize mathematically, and which are therefore not conducive to classical analysis based engineering tools.

Acknowledgements

The authors would like to acknowledge the support of the Cooperative Research Center (VIKKK) (KKK-I-7), the Hungarian Ministry of Education (FKFP-0063/2000 and FKFP-0073/2001), and the Hungarian Science Foundation (OTKA TO37600).

References

[1] McMullin, B, "*John von Neumann and the Evolutionary Grwoth of Complexity: Looking Backwards, Looking Forwards…*", Artificial Life, Vol.6, Issue 4, pp. 347-361, 2000.

[2] http://c3.lanl.gov/~rocha/ss504_5.html

[3] Rechenberg, I., "*Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution*", Stuttgart: Frommann-Holzboog, 1973.

[4] Holland, J.H., "*Adaptation in Natural and Artificial Systems*", Ann Arbor, Michigan: The University of Michigan Press, 1975.

[5] Spears, W.M., De Jong, K.A., Back, T., Fogel, D.B. and Garis H. "*An Overview of Evolutionary Computation*", European Conference on Machine Learning, 1993.

[6] Mandischer, M., "*A comparison of evolutionary strategies and backpropagation for neural network training*", Neurocomputing, Vol. 42, 2002, pp. 87-117.

[7] Fogel, L.J., Owens, A.J., and Walsh, M.J., "*Artificial Intelligence Through Simulated Evolution*", New York: Wiley Publishing, 1966.

[8] http://web.ics.purdue.edu/~hux/tutorials.shtml

[9] Andres-Toro, B. de, Giron-Sierra, J.M., Lopez-Orozco, J.A., Fernandez-Conde, C., Peinado, J.M. and Garcia-Ochoa, F., "*A kinetic model for beer production under industrial operational conditions*", Mathematics and Computers in Simulation, Vol. 48, 99.65-74, 1998

[10] Carrillo-Uerta, G.E., Roberts, P.D. and Becerra, P.D., "*Genetic algorithms for optimal control of beer fermentation*", Proc. IEEE International Symposium on Intelligent Control, Mexico City, Mexico, 391-396, 2001

[11] http://www.automaton.ch/ca/vonNeumann.html

[12] Luis Rocha, http://www.c3.lanl.gov/~rocha/ss504_67.html#L7

[13] Ingber, L. and Rosen, B., "*Genetic Algorithms and Very Fast Simulated Reannealing: A Comparison*", Mathematical and Computer Modelling, Vol. 16(11), pp. 87-100, 1992.