An Application of Genetic Algorithms to Membrane Computing

Gabi Escuela[1], Miguel A. Gutiérrez-Naranjo[2]

[1] Bio Systems Analysis Group
Friedrich Schiller University Jena
gabi.escuela@uni-jena.de
[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
magutier@us.es

Summary. The process of designing a P system in order to perform a task is a hard job. The researcher has often only an approximate idea of the design, but finding the exact description of the rules is a heavy hand-made work. In this paper we introduce *PSystemEvolver*, an evolutionary algorithm based on generative encoding, that could help to design a P system to perform a specific task. We illustrate the use of *PSystemEvolver* with a simple mathematical problem: the computation of squared numbers.

1 Introduction

Natural Computing studies computational paradigms inspired from various well known natural phenomena in physics, chemistry and biology[3]. It abstracts the way in which nature computes, conceiving new computing models. The field is growing rapidly and there are many open research lines based on different aspects in which nature acts. Among them, *Cellular Automata* [16] conceived by Ulam and von Newman as a spatial distribution of cells able to reproduce the behavior of complex systems; *Genetic algorithms* introduced by J. Holland [13] which is inspired by natural evolution and selection in order to find a good solution in a large set of feasible candidate solutions; *Neural Networks* introduced by W.S. McCulloch and W. Pitts [15] it is based on the interconnections of neurons in the brain; *DNA-based* molecular computing, that was born when L. Adleman [2] published a solution to an instance of the Hamiltonian path problem by manipulating DNA strands in a lab; *Swarm Intelligence* [6] based on the behavior of mobile organisms as ants or bees communicating among them and acting in the environment; *Artificial Immune Systems* [5] based on the natural immune system

[3] An introduction on Natural Computing can be found in [12].

of biological organisms; *Amorphous computing* [1] inspired from the development of morphogenesis in biological organisms or *Membrane Computing* [17, 18] based on the functioning and morphology of living cells and tissues.

Membrane Computing was introduced by Gh. Păun in [17] under the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations. The devices of this model are called *P systems*. Roughly speaking, a P system consists of a membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules in a synchronous nondeterministic maximally parallel manner.

The basic idea is to consider a distributed and parallel computing device structured in an arrangement of membranes which delimit compartments where various chemicals evolve according to local reaction rules. The objects can be eventually sent to the environment or to adjacent membranes under the control of specific rules. Because the chemicals from the compartments of a cell are swimming in an aqueous solution, the data structure we consider is that of a *multiset* – a set with multiplicities associated with its elements. Also, in close analogy with what happens in a cell, the reaction rules are applied in a parallel manner, with the objects to evolve by them and with the reactions themselves chosen in a non–deterministic manner.

In this way, we can define transitions from a configuration to another configuration of our system and hence we can define computations. A computation provides a result, for instance, in the form of the number of objects present in the halting configuration in a specified compartment, or in the form of a special object, `yes` or `no`, sent to the environment at the end of the computation (thus answering a decision problem that the system had to solve).

Evolutionary Algorithms (EAs) are generic population-based metaheuristics inspired in biological evolution to deal with combinatorial optimization problems. Four main EAs have been applied to different kind of problem domains: Genetic Algorithms, Genetic Programming, Evolutionary strategies and Evolutionary programming. Genetic Algorithms were introduced by J.H. Holland [13] an American psychologist and computer scientist who developed his theory to study self-adaptiveness in biological processes as well as to solve optimization problems. Concerning to functioning, a genetic algorithm is an iterative procedure which operates on a population where individuals are evaluated according a certain fitness value. Some individuals are selected according this value and produce offspring candidates which form the next generation[4]. For producing new individuals, two operators, namely crossover and mutation are used. Crossover takes two individuals called parents and produce one or two new individuals called offsprings. In its simplest form, it works by swapping pieces of information from the parents. The second operator is called mutation and it is applied by modifying an information unit in one individual according to a mutation rate.

In this paper we present a case study where genetic algorithms are used for designing a Membrane Computing device which performs a pre-fixed task. In the

[4] For details, see, for example [3].

literature, one can find several joint approaches of Membrane Computing and Genetic Algorithms as [14] or [4], but, to the best of our knowledge, this is the first time in which genetic algorithms are used to find a P system which solves an abstract computational problem.

The paper is organized as follows: Next we describe our case study for the application of Genetic Algorithms to the design of P systems. We start by describing an initial *population* of P systems and the genetic operators to act on them. In Section 3 we provide a short description of the genetic algorithm *PSystemEvolver* used in our experiments. In the following sections we provide the obtained experimental results. The paper ends with some final remarks and open lines for further research.

2 The Problem

The process of designing a P system in order to perform a task is a hard job. In many cases, the designer has an approximate idea of the membrane structure, initial multisets and set of rules necessary to describe the P system, but a little mistake in the description of the initial configuration or in the set of rules can leads to undesired consequences.

In this paper we present the case study of designing a P system which computes the square of a given number, e.g., number 4. To this aim, we will consider an initial *population* of P systems. Such population will *evolve* according to the natural selection of the evolution of alive beings (by means the corresponding *crossover* and *mutation* operations of a given genetic algorithm) and with the help of a *fitness* function we will obtain a member of the *population*, i.e., a P system obtained from the original ones, which performs the fixed task.

In order to perform our experiments, we consider that all the P systems have the same initial configuration. The allowed set of rules are of the types:

- **Evolution rules:** $[\, o \rightarrow u \,]_e$. The object o evolves to the multiset u in membrane with label e. Notice that u can be the *empty multiset* λ.
- **Dissolution rules:** $[\, r \,]_e \rightarrow s$. The object r dissolves the membrane e and goes to the surrounding region as object s. All the remaining objects in e also go to the surrounding region.

Our starting point is to consider a family of P systems $\mathbf{\Pi} = \{\Pi_i\}_{i \in I}$ where

$$\Pi = \langle \Gamma, H, \mu, w_e, w_s, R_i \rangle$$

- The alphabet $\Gamma = \{a, b, c, z_1, \ldots, z_4\}$
- The set of labels $H = \{e, s\}$
- The membrane structure $\mu = [\,[\,]_e\,]_s$
- The initial multisets $w_e = a^2\, b\, z_1$ and $w_s = \emptyset$

The difference among the P systems in the family are the sets of rules R_i. In order to give a formal definition of $\mathbf{\Pi} = \{\Pi_i\}_{i \in I}$ we will start with a set of rules $\mathcal{R}$

$$\mathcal{R} = \left\{ \begin{array}{lll} r_1 \equiv [\, a \rightarrow a\, b\,]_e & r_7 \equiv [\, z_2 \rightarrow z_1\,]_e & r_{13} \equiv [\, a \rightarrow \lambda\,]_s \\ r_2 \equiv [\, b \rightarrow b\, c\,]_e & r_8 \equiv [\, z_3 \rightarrow z_4\,]_e & r_{14} \equiv [\, b \rightarrow \lambda\,]_s \\ r_3 \equiv [\, c \rightarrow b^2\,]_e & r_9 \equiv [\, z_1\,]_e \rightarrow b & r_{15} \equiv [\, b \rightarrow c\,]_e \\ r_4 \equiv [\, a \rightarrow b\, c\,]_e & r_{10} \equiv [\, z_2\,]_e \rightarrow a & r_{16} \equiv [\, c \rightarrow \lambda\,]_e \\ r_5 \equiv [\, z_1 \rightarrow z_2\,]_e & r_{11} \equiv [\, z_3\,]_e \rightarrow c & r_{17} \equiv [\, z_4 \rightarrow z_1\,]_e \\ r_6 \equiv [\, z_2 \rightarrow z_3\,]_e & r_{12} \equiv [\, z_4\,]_e \rightarrow a & r_{18} \equiv [\, z_4\,]_e \rightarrow b \end{array} \right\}$$

Our aim is to use genetic algorithms in order to find a P system which computes the square of number 4, from an initial set of P systems. The genetic evolution will only correspond to changes in the set of rules. The genetic operations in order to develop a genetic algorithm on the P systems are the following:

- **Crossover.** Given two P systems Π_1 and Π_2 and their sets of rules R_1 and R_2, let P_1^1 P_1^2 and P_2^1 P_2^2 two partitions of R_1 and R_2 respectively. Then, we obtain two offsprings Π_1' and Π_2' by considering the set of rules $R_1' = P_1^1 \cup P_2^1$ and $R_2' = P_1^2 \cup P_2^2$.
- **Mutation.** Given an evolution rule $[u \rightarrow v]_h$ with $u \in \Gamma$ and $v \in \Gamma^*$, the mutation operator changes the object u by one from $\Gamma - \{u\}$ or the object w in the multiset v by one object from $\Gamma - \{w\}$ or by λ. For an dissolution rule $[u]_h \rightarrow w$, the mutation operator changes the object u or w by a different one from Γ.

Only for practical reasons, in this case study we will impose an extra condition. All the P systems considered as individuals in our genetic algorithm must be deterministic. This is checked by ensuring that, for each P system and each membrane, there are no two rules triggered by the same object.

Example 1. Let us consider two P systems Π_1 and Π_2 and their sets of rules $R_1 = \{r_1^1, r_1^2\}$ and $R_2 = \{r_2^1, r_2^2, r_2^3\}$ with

$$\begin{array}{ll} r_1^1 \equiv [\, a \rightarrow a\, b\,]_e & r_2^1 \equiv [\, a \rightarrow b\, c\,]_e \\ r_1^2 \equiv [\, c \rightarrow b^2\,]_e & r_2^2 \equiv [\, z_4 \rightarrow z_1\,]_e \\ & r_2^3 \equiv [\, z_1\,]_e \rightarrow b \end{array}$$

Let P_1^1 and P_1^2 be a partition of R_1, $P_1^1 = \{r_1^1\}$ and $P_1^2 = \{r_1^2\}$ and P_2^1, P_2^2 a partition of R_2 with $P_2^1 = \{r_2^2, r_2^3\}$ and $P_2^2 = \{r_2^1\}$. Then, we obtain two offsprings Π_1' and Π_2' by considering the set of rules $R_1' = P_1^1 \cup P_2^1 = \{r_1^1, r_2^2, r_2^3\}$ and $R_2' = P_1^2 \cup P_2^2 = \{r_1^2, r_2^1\}$. Notice that, due to the restriction of determinism, we cannot get a new offspring by joining P_1^1 and P_2^2.

As example of the mutation operator, let us consider now the rule $[\, a \rightarrow b\, c\,]_e$. By applying a mutation rule we can obtain, for example, the rules $[\, z_1 \rightarrow b\, c\,]_e$ (changing a by z_1), the rule $[\, a \rightarrow b^2\,]_e$ (changing c by b) or $[\, a \rightarrow c\,]_e$ (changing b by λ).

Finally, we can describe the set of P systems $\mathbf{\Pi}$ considered as individuals for our genetic algorithm. A P system Π belongs to $\mathbf{\Pi}$ if it is a construct $\langle \Gamma, H, \mu, w_e, w_s, R_i \rangle$ as described above and the rules in R_i are from $\mathcal{R}$ or can be derived by a finite number of applications of the operators *crossover* and *mutation* from rules in $\mathcal{R}$.

3 The Genetic Algorithm

In this section we will briefly describe the genetic algorithm, *PSystemEvolver*, used in our case study. It follows the basic workflow:

Produce an initial population of individuals
Evaluate the fitness of all individuals
while termination condition not met **do**
  Select the best individuals and produce new individuals (crossover and mutation operators)
  Evaluate the fitness of new individuals
  Generate the new population inserting the best individual from previous generations
end while

In order to apply the previous algorithm, we need to precise some details:

- The initial population consists on 30 individuals. In order to generate these individuals, 30 different random subsets of $\mathcal{R}$ are considered. The number of rules of each individual will not exceed 14, that is, the length of the alphabet times the number of membranes that we are considering. Before evaluating a possible solution using the fitness function, the P system is checked to assure determinism. If more than one rule in a specific membrane has the same right hand side (firing object), one of them is selected randomly to be active and the others are deleted from this P system.
- The fitness function is probably the key point in the application of the genetic algorithm for the design of P systems. In this case study we have considered a simple function: The absolute value of the difference between the number of objects c in the membrane s in the halting configuration of the P system and the expected number of such objects in an ideal found solution, i.e., 16 objects c. In order to prevent non-ending computations, we we limit to 20 the number of steps.
- The crossover and mutation operators will be applied on some randomized individuals with *good* score in the fitness function. For that, two parents are selected according to their fitness and mated to produce two offsprings that later could be mutated. Crossover and mutation rates of 0.8 and 0.8, respectively, have been considered in order to perform our experiments. We also varied this parameters to test the effect of this operators over the performance of the algorithm.

- As termination condition for the algorithm, a maximum of 30 generations has been considered.

4 Experimental Results

The chosen fitness function for our experiments with *PSystemEvolver* evaluates each P system according to its halting configuration. The computer simulations of all the computations have been performed by using the P-lingua simulator [9]. To calculate the fitness of each individual, *PSystemEvolver* generates the corresponding P-lingua file and call with it the simulator that produces a report file to obtain the evaluation for that P system.

To test the behavior of the algorithm, we performed 30 runs for each EA parameters setting. Table 1 shows the number of success for each experiment, that is, the number of times that *PSystemEvolver* could find a P system that solve the square(4) problem.

Experiment	**Crossover Rate**	**Mutation Rate**	**Successful Runs**
1	0.0	0.5	0/30
2	0.5	0.5	0/30
3	0.8	0.8	1/30
4	1.0	0.8	1/30

Table 1. Number of successful runs for different parameter settings.

Results demonstrated that is difficult to evolve a P system, even though initial configuration and membrane structure are fixed, and rules are provided for generating the initial population. This may be due to the fitness function that we considered for this problem, that conforms a landscape with a unique peak.

High values for crossover and mutation rates resulted beneficial for this algorithm, as can be seen in 1. Other types of mutations, as for example, rule activation or inactivation would be considered in future implementations.

The best P system P_{best} encountered by *PSystemEvolver* is described above by the rules R_{best}:

$$R_{best} = \left\{ \begin{array}{ll} r_1 \equiv [\, a \rightarrow a\, b\,]_e & r_5 \equiv [\, z_3 \rightarrow z_4\,]_e \\ r_2 \equiv [\, b \rightarrow b\, c\,]_e & r_6 \equiv [\, z_4\,]_e \rightarrow a \\ r_3 \equiv [\, z_1 \rightarrow z_2\,]_e & r_7 \equiv [\, a \rightarrow \lambda\,]_s \\ r_4 \equiv [\, z_2 \rightarrow z_3\,]_e & r_8 \equiv [\, b \rightarrow \lambda\,]_s \end{array} \right\}$$

5 Final Remarks

The advances in the research in Membrane Computing requires the design of more and more complex P systems. On the one hand, the theoretical research needs

sophisticated designs which allows prove the ability of P systems for solving different type of problems by using a fixed ingredients (see, e.g., [10, 11]). On the other hand, Membrane Computing solutions to real-life problems needs to be quite precise in the design in order to find a sharp simulation of the processes [7, 8].

The design of such P systems is a hard task which must be performed by hand by the researcher. In this paper we explore the use of Genetic Algorithms as a help for designing P systems. The key point is finding a good fitness function. P systems are designed to make a computation and it is difficult to measure how far is the current design from the desired when the result of the computation is not the searched.

Many open questions arise from this work. As pointed above, the problem of finding the features of a *good* fitness function is open, but this is not the only one. A first research line involves a deeper study of the genetic algorithm operators, not only for the fitness function, but operators able also to modify the initial multisets or the membrane structures. A second line is related to the applications. In this paper we use a small theoretical problem, but the final target is to apply genetic algorithms for the design of complex P systems.

Acknowledgement

GE is supported by Universidad Simón Bolívar (Venezuela) and Deutscher Akademischer Austausch Dienst (DAAD) Grant A/08/94489.
MAGN acknowledges the support of the projects TIN2008-04487-E and TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight Jr., R. Nagpal, E. Rauch, G. Sussman, and R. Weiss. Amorphous computing. *Communications of the ACM* **43**(5), (2000) 74-82.
2. L.M. Adleman. Molecular computations of solutions to combinatorial problems. *Science*, **226** (1994) 1021–1024.
3. M. Affenzeller, S. Winkler, S. Wagner, A. Beham. Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications. Chapman & Hall/CRC. (2009).
4. H. Cao, F.J. Romero-Campero, S. Heeb, M. Cámara, N. Krasnogor. Evolving cell models for systems and synthetic biology. *Systems and Synthetic Biology* **4**(1), (2010) 55–84.
5. L. de Castro and J. Timmis. Artificial Immune Systems: A New Computational Intelligence Approach. Springer, (2002).
6. A. Engelbrecht. Fundamentals of Computational Swarm Intelligence. Wiley and Sons, (2005).

7. G. Escuela, T. Hinze, P. Dittrich, S.Schuster, M. Moreno. Modelling Modified Atmosphere Packaging for Fruits and Vegetables using Membrane Systems. Accepted paper at Third International Conference on Bio-inspired Systems and Signal Processing BIOSIGNALS 2010. Valencia, Spain. January 2010.
8. T. Hinze, T. Lenser, G. Escuela, I. Heiland, S. Schuster. Modelling Signalling Networks with Incomplete Information about Protein Activation States: A P System Framework of the KaiABC Oscillator. *Lecture Notes in Computer Science* **5957**, (2010), 316-334.
9. M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. An overview of P-Lingua 2.0. *Lecture Notes in Computer Science*, **5957** (2010), 264-288.
10. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez. Solving Subset Sum in linear time by using tissue P systems with cell division. *Lecture Notes in Computer Science* **4527** (2007), 170-179.
11. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, F.J. Romero-Campero. Computational efficiency of dissolution rules in membrane systems. *International Journal of Computer Mathematics* **83**(7), (2006) 593 - 611.
12. L. Kari and G. Rozenberg. The many facets of Natural Computing. *Communications of the ACM*, **51**(10), (2008) 72–83.
13. J.H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press. (1975)
14. L. Huang and N. Wang. An Optimization Algorithm Inspired by Membrane Computing. *Lecture Notes in Computer Science* **4222**, (2006) 49-52.
15. W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* **5** (1943) 115–133.
16. J. von Neumann. Theory of Self-Reproducing Automata. U. Illinois Press (1966).
17. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143.
18. Gh. Păun. *Membrane Computing – An Introduction* Springer-Verlag, Berlin, 2002.