Distributed Evolutionary Algorithms Inspired by Membranes in Solving Continuous Optimization Problems

Daniela Zaharie[1] and Gabriel Ciobanu[2]

[1] Department of Computer Science, West University of Timişoara
Blvd. V. Pârvan no.4, 300223 Timişoara, Romania
dzaharie@info.uvt.ro
[2] Institute of Computer Science, Romanian Academy
Blvd. Carol I no.8, 700505 Iaşi, Romania
gabriel@iit.tuiasi.ro

Abstract. In this paper we present a new strategy to apply the operators in evolutionary algorithms. This strategy is inspired from the way the evolution rules are used in membrane systems. Moreover, analyzing the similarities and differences between the evolutionary operators and the evolution rules in membrane systems, between membrane structures and communication topologies, and between communication rules in membrane systems and communication policies in evolutionary algorithms, we introduce and a hybrid distributed evolutionary algorithm and test it for continuous optimization problems.

1 Introduction

Membrane systems and evolutionary algorithms are computation models inspired by nature, both based on applying some evolution(ary) rules to a (multi)set of simple or structured objects. Both models have distributed features. Membrane systems represent a suitable framework for distributed algorithms [2], and evolutionary algorithms allow natural extensions for distributed implementation [12].

A *membrane system* consists of a hierarchy of membranes that do not intersect, with a distinguishable membrane, called the *skin membrane*, surrounding them all. A membrane without any other membranes inside is *elementary*, while a non-elementary membrane is a *composite* membrane. The membranes produce a demarcation between *regions*. For each membrane there is a unique associated region. The space outside the skin membrane is called the environment. Because of this one-to-one correspondence we sometimes use membrane instead of region. Regions contain multisets of *objects*, *evolution rules* and possibly other membranes. Only rules in a region delimited by a membrane act on the objects in that region. The multisets of objects from a region correspond to the "chemicals swimming in the solution in the cell compartment", while the rules correspond to the "chemical reactions possible in the same compartment". The rules must

contain target indications, specifying the membrane where the new objects obtained after applying the rule are sent. The new objects either remain in the same region when they have a *here* target, or they pass through membranes, in two directions: they can be sent *out* of the membrane delimiting a region from outside, or can be sent *in* one of the membranes delimiting a region from inside, precisely identified by its label. In a step, the objects can pass only through one membrane. There exist many variants and classes of membrane systems; many of them are introduced in [7].

Evolutionary algorithms are reliable in solving hard problems in the field of discrete and continuous optimization. They are approximation algorithms which achieve a trade-off between solution quality and computational costs. Despite the large variety of evolutionary algorithms (genetic algorithms, evolution strategies, genetic programming, evolutionary programming), all of them are based on the same idea: evolve a *population* of candidate solutions by applying some rules inspired by biological evolution: *recombination (crossover)*, *mutation*, and *selection* [3]. An evolutionary algorithm acting on only one population is similar to a one-membrane system. *Distributed evolutionary algorithms* which evolve separate but communicating (sub)populations are more like membrane systems.

It is natural to ask questions as: *How similar are membrane computing and distributed evolutionary computing? Can ideas from membrane computing improve the evolutionary algorithms, or vice-versa?* An attempt to build a bridge between membrane computing and evolutionary computing is given in [6], where a membrane algorithm is developed by using a membrane structure together with ideas from genetic algorithms (crossover and mutation operators) and from metaheuristics for local search (tabu search). In this paper we go further and deeper, and analyze the relationship between different membranes structures and different communication topologies specific to distributed evolutionary algorithms. Moreover, we develop a hybrid distributed evolutionary algorithm for continuous optimization characterized through a non-standard, membrane systems inspired, strategy for applying the evolutionary operators.

The paper is organized as follows. In Section 2 we analyze the correspondence between evolutionary operators and evolution rules. As a result of this analysis, we propose a new strategy of applying the evolutionary operators. Section 3 is devoted to the similarities between membrane structures and communication topologies on one hand, and between communication rules in membrane systems and communication policies in evolutionary algorithms on the other hand. In Section 4 we introduce a new hybrid distributed evolutionary algorithm, and we test its effectiveness in solving continuous optimization problems.

2 Evolutionary Operators and Evolution Rules

Evolutionary algorithms (EAs) working on only one population (panmictic EAs) can be interpreted as particular membrane systems having only one membrane. Inside this single membrane there is a population of candidate solutions for the problem to be solved. Usually a population is an m-uple of n-dimensional

vectors: $P = x_1 \dots x_m$, $x_i = (x_i^1, \dots, x_i^n) \in D$, where D is a discrete or a continuous domain depending on the problem to be solved. The evolutionary process consists in applying the recombination, mutation and selection operators to the current population in order to obtain a new population.

Recombination (crossover): The aim of this operator is to generate new elements from a set of elements (called parents) selected from the current population. Thus we have a mapping $\mathcal{R} : D^r \to D^q$, where usually $q \leq r$. Typical examples of recombination operators are:

$$r = q = 2, \qquad \mathcal{R}((u^1, \dots, u^n), (v^1, \dots, v^n)) = ((u^1, \dots, u^k, v^{k+1}, \dots, v^n), (v^1, \dots, v^k, u^{k+1}, \dots, u^n)) \tag{1}$$

and

$$r \text{ arbitrary }, q = 1, \qquad \mathcal{R}(x_{i_1}, \dots, x_{i_r}) = \frac{1}{r} \sum_{j=1}^{r} x_{i_j} \tag{2}$$

The first type of mutation corresponds to one point crossover (where $k \in \{1, \dots, n-1\}$ is an arbitrary cut point) used in genetic algorithms, while the second example corresponds to intermediate recombination used in evolution strategies [3].

Mutation: The aim of this operator is to generate a new element by perturbing one element from the current population. Thus we work with a mapping $\mathcal{M} : D \to D$ defined by $\mathcal{M}((u^1, \dots, u^n)) = (v^1, \dots, v^n)$. Typical examples are:

$$v^i = \begin{cases} 1 - u^i & \text{with probability } p_m \\ u^i & \text{with probability } 1 - p_m \end{cases} \qquad \text{and} \qquad v^i = u^i + N(0, \sigma^i), \quad i = \overline{1, n} \tag{3}$$

The first example is used in genetic algorithms based on a binary coding ($u^i \in \{0, 1\}$), while the second one is typical for evolution strategies. $N(0, \sigma^i)$ denotes a random variable with normal distribution, of zero mean and standard deviation σ^i.

Selection: It is used to construct a new set of elements starting from the current population such that the best elements with respect to the objective function of the optimization problem to be solved are favored. It does not generate new configurations, but only sets of existing (not necessarily distinct) configurations. Thus it maps D^m to D^r and can be used in two main situations: selection of parents for recombination (in this case $r < m$, and the parents selection is not necessarily based on the quality of elements), and selection of survivors (in this case $r = m$, and the survivors are stochastically or deterministically selected by taking into account their quality with respect to the optimization problem). In the following, the mapping corresponding to parents selection is denoted by $\mathcal{S}_p$ and the mapping corresponding to survivors selection is denoted by $\mathcal{S}_s$.

A particular evolutionary algorithm is obtained by combining these evolutionary operators and by applying them iteratively to a population. Typical ways

of combining the evolutionary operators lead to the main evolutionary strategies: generational, and steady state. In the generational (synchronous) strategy, at each step a population of new elements is generated by applying recombination and mutation. The population of the next generation is obtained by applying selection to the populations of new and old elements. The general structure of a generational EA is presented in Algorithm 1 where $X(t)$ denotes the population corresponding to generation t and Z denotes the population of offsprings. The symbol $\cup_+$ denotes an extended union, which allows multiple copies of the same element (as in multisets). The mapping $\overline{\mathcal{M}}$ is the extension of $\mathcal{M}$ to D^q, i.e. $\overline{\mathcal{M}}(x_{i_1}, \ldots, x_{i_q}) = (\mathcal{M}(x_{i_1}), \ldots, \mathcal{M}(x_{i_q}))$.

Algorithm 1 Generational Evolutionary Algorithm

1: Random initialization of population $X(0)$
2: $t := 0$
3: **repeat**
4: $\quad Z := \emptyset$
5: $\quad$ **for all** $i \in \{1, \ldots, m\}$ **do**
6: $\quad\quad Z := Z \cup_+ (\overline{\mathcal{M}} \circ \mathcal{R} \circ \mathcal{S}_p)(X(t))$
7: $\quad$ **end for**
8: $\quad X(t+1) := \mathcal{S}_s(X(t) \cup_+ Z)$
9: $\quad t := t + 1$
10: **until** a stopping condition is satisfied

In the steady state (asynchronous) strategy, at each step a new element is generated by recombination and mutation, and assimilated into the population if it is good enough (e.g. better than one of its parents, or than the worst element in the population). More details are in Algorithm 2.

Algorithm 2 Steady State Evolutionary Algorithm

1: Random initialization of population $X(0)$
2: $t := 0$
3: **repeat**
4: $\quad z := (\overline{\mathcal{M}} \circ \mathcal{R} \circ \mathcal{S}_p)(X(t))$
5: $\quad X(t+1) := \mathcal{S}_s(X(t) \cup_+ z)$
6: $\quad t := t + 1$
7: **until** a stopping condition is satisfied

The simplest way to present a generational or a steady state evolutionary algorithm as a membrane system is to consider the entire population as a structured object in a membrane, and the compound operator applied as one evolution rule which includes recombination, mutation and selection. Such an approach represents a rough and coarse handling which does not offer flexibility. A more

flexible approach would be to consider each evolutionary operator as an evolution rule.

Evolutionary operators are usually applied in an ordered manner (as in Algorithms 1 and 2): first parents selection, then recombination and mutation, and finally survivors selection. Starting from the way the evolution rules are applied in a membrane system, we consider that *the rules can be independently applied to the population elements*, meaning that no predefined order between the operators is imposed. At each step any operator can be applied, up to some restrictions ensuring the existence of the population. The recombination and mutation operators $\mathcal{R}$ and $\mathcal{M}$ can be of any type, with possible restrictions imposed by the coding variant. By applying these operators, new elements are created. These elements are unconditionally added to the population. Therefore by applying the recombination and mutation operators, the population size is increased. When the population size reaches an upper limit (e.g. twice the size of the initial population), then $\mathcal{R}$ and $\mathcal{M}$ are inhibited.

The role of the selection is to modify the distribution of elements in the population by eliminating or by cloning some elements. A simple selection operator could be defined by eliminating the worst element of the population, or by cloning the best element of the population. When selection is applied by cloning, then the population size is increased and selection is inhibited whenever the size reaches a given upper bound. On the other hand, when selection is applied by eliminating the worst element, the population size is reduced, and selection is inhibited whenever the size reaches a given lower bound.

By denoting with $x_1 \dots x_m$ ($x_i \in D$) the entire population, with $x_{i_1} \dots x_{i_q}$ an arbitrary part of the population, with x_* the best element and with x_- the worst element, the evolutionary operators can be described more in the spirit of evolution rules from membrane systems as follows:

Rule 1 (recombination): $x_{i_1} \dots x_{i_r} \to x_{i_1} \dots x_{i_r} x'_{i_1} \dots x'_{i_q}$ where $(x_{i_1}, \dots, x_{i_r}) = \mathcal{S}_p(x_1, \dots, x_m)$ is the set of parents defined by the selection operator $\mathcal{S}_p$, and $(x'_{i_1}, \dots, x'_{i_q}) = \mathcal{R}(x_{i_1}, \dots, x_{i_r})$ is the offspring set obtained by applying the recombination operator $\mathcal{R}$ to this set of parents;

Rule 2 (mutation): $x_i \to x_i x'_i$ where $x'_i = \mathcal{M}(x_i)$ is the perturbed element obtained by applying the mutation operator $\mathcal{M}$ to x_i;

Rule 3a (selection by deletion): $x_- \to \lambda$, meaning that the worst element (with respect to the objective function) is eliminated from the population;

Rule 3b (selection by cloning): $x_* \to x_* x_*$ meaning that the best element (with respect to the objective function) is duplicated.

By following the spirit of membrane computing, these rules should be applied in a fully parallel manner. However, in order to avoid going too far from the classical way of applying the operators in evolutionary algorithms, we consider a sequential application of rules. Thus we obtain an intermediate strategy: the evolutionary operators are applied sequentially, but in an arbitrary order. Such a strategy (which uses a deletion type selection) based on a probabilistic decision concerning the operator to be applied at each step is described in Algorithm

3, where p_R, p_M, $p_S \in (0,1)$ are the probabilities of applying recombination, mutation and selection, respectively and satisfy $p_R + p_M + p_S = 1$. By applying the evolutionary operators in such a fully asynchronous way, we obtain a more flexible algorithm which works with variable size populations. In Algorithm 3 the population size corresponding to iteration t is denoted by $m(t)$.

We can expect that the behaviour of such an algorithm be different from the behaviour of more classical generational and steady state algorithms. However, from a theoretical viewpoint, such an algorithm can be still modelled by a Markov chain and the convergence results still hold [10]. This means that if we use a mutation operator based on a stochastic perturbation described by a distribution having a support which covers the domain D (e.g. normal distribution) and an elitist selection (the best element found during the search is not eliminated from the population), then the best element of the population converges in probability to the optimum.

Algorithm 3 Fully Asynchronous Evolutionary Algorithm

Initialize the population $X(0) = x_1(0) \dots x_{m(0)}(0)$
$t := 0$
repeat
  generate a uniform random value $u \in (0,1)$
  if $(u < p_R) \wedge (m(t) < 2m(0))$ **then**
    apply *R1 (recombination)*
  end if
  if $(u \in [p_R, p_R + p_M)) \wedge (m(t) < 2m(0))$ **then**
    apply *R2 (mutation)*
  end if
  if $(u \in [p_R + p_M, 1]) \wedge (m(t) > m(0)/2)$ **then**
    apply *R3a (selection by deletion)*
  end if
  $t := t + 1$
until a stopping condition is satisfied

The difference appears with respect to the finite time behavior of the algorithm, namely the ability to approximate (within a certain desired precision) the optimum in a finite number of steps. Preliminary tests suggest that for some optimization problems, the fully asynchronous strategy works better than the generational and steady state strategies; numerical results are presented in Section 4. This means that using ideas from the application of evolution rules in membrane systems, we can obtain new evolutionary strategies with different dynamics.

3 Communication Topologies and Policies

As it has been stated in the previous section, a one-population evolutionary algorithm can be mapped into a one-membrane system with rules associated to

the evolutionary operators. Closer to membrane computing are the distributed evolutionary algorithms which work with multiple (sub)populations. In each subpopulation the same or different evolutionary operators can be applied leading to homogeneous or heterogeneous distributed EAs, respectively. Introducing a structure over the population has different motivations [12]: (i) it achieves a good balance between exploration and exploitation in the evolutionary process in order to prevent premature convergence (convergence to local optima) in the case of global optimization problems; (ii) it stimulates the population diversity in order to deal with multimodal optimization problems or with dynamic optimization problems; (iii) it is more suitable to parallel implementation.

Therefore, besides the possibility of improving the efficiency by parallel implementation, structuring the population in communicating subpopulations allows developing new search mechanisms which behave differently than their serial counterparts [12]. The *multi-population model of the evolutionary algorithms*, also called *island-model*, is based on the idea of dividing the population in some communicating subpopulations. In each subpopulation is applied an evolutionary algorithm for a given number of generations, then a migration process is started. During the migration process some elements can change their subpopulations, or clones of some elements can replace elements belonging to other subpopulations. The main elements which influence the behaviour of a multi-population evolutionary algorithm are the *communication topology* and the *communication policy*. The communication topology specifies which subpopulations are allowed to communicate while the communication policy describes how is ensured the communication. The communication topology in a distributed evolutionary algorithm plays a similar role as the membranes structure plays in a membrane system. On the other hand the communication policy in distributed evolutionary algorithms is related to the communication rules in membrane systems.

3.1 Communication Topologies and Membrane Structures

The communication topology describes the connections between subpopulations. It can be modelled by a graph having nodes corresponding to subpopulations, and edges linking subpopulations which communicate in a direct manner. According to [1], typical examples of communication topologies are: fully connected topology (each subpopulation can communicate with any other subpopulation), linear or ring topology (only neighbour subpopulations can communicate), star topology (all subpopulations communicate through a kernel subpopulation). More specialized communication topologies are hierarchical topologies [5], and hypercube topologies [4]. The fully connected, star and linear topology can be easily described by using hierarchical membrane structures which allows transferring element either in a inner or in the outer membrane (see Figure 1).

(a) *Fully connected topology:* Let us consider a number of s fully connected subpopulations. The fully connected topology can be modelled by using $s + 1$ membranes, namely s elementary membranes and one skin membrane containing them (see Figure 1(a)). The elementary membranes correspond to

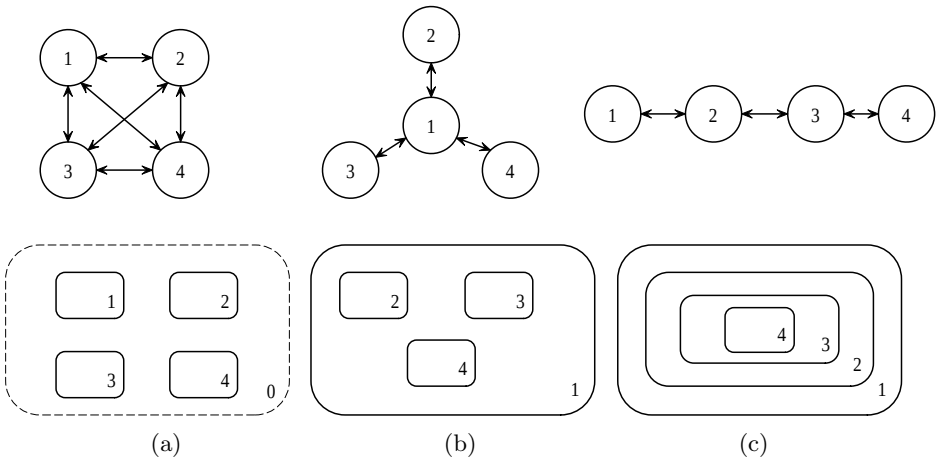

Fig. 1. Communication topologies in distributed evolutionary algorithms and their corresponding membranes structures. (a) Fully connected topology; (b) Star topology; (c) Linear topology.

the given s subpopulations, and they contain both evolution rules and communication rules. The skin membrane plays only the role of communication environment, thus it contains only communication rules and the objects which have been transferred from the inner membranes. The transfer of an element between two inner membranes is based on two steps: the transfer of the element from the source membrane to the skin membrane and the transfer of the element from the skin membrane to the target membrane. Another structure which corresponds to a fully connected topology is that associated to tissue P-systems.

(b) *Star topology:* The membrane structure corresponding to a star topology with s subpopulations is given by one skin membrane corresponding to the kernel subpopulation, and $s - 1$ elementary membranes corresponding to the other subpopulations (see Figure 1(b)). The main difference from the previous structure associated to a fully connected topology is that the skin membrane has not only the role of an environment for communication, but it contains also some evolution rules.

(c) *Linear topology:* In this case a subpopulation p can communicate only with its neighbour subpopulations $p + 1$ and $p - 1$. The corresponding structure is given by nested membranes, each membrane corresponding to a subpopulation (see Figure 1(c)).

Different situations appear in the case of ring and other topologies [4] which are associated with cyclic graph structures. In these situations the corresponding membrane structure is given by a net of membranes, or tissue P systems.

3.2 Communication Policies and Communication Rules

A communication policy refers to the way the communication is initiated, the way the migrants are selected, and the way the immigrants are incorporated into the target subpopulation. Communication can be initiated in a synchronous way after a given number of generations, or in an asynchronous way when a given event occurs. The classical variants of migrants selection are random selection and selection based on the fitness value (best elements migrate and the immigrants replace the worst elements of the target subpopulation). The communication policies are similar to communication rules in membrane computing, meaning that all communication steps can be described by some typical communication rules in membrane systems.

Example: Let us consider the communication by random migration in the case of a fully connected topology (Figure 1(a)). This means that whenever a migration step is initiated, any element from a subpopulation S_i can be selected (with a given probability) to migrate to a (also randomly selected) target subpopulation S_j.

There are two main variants for transferring elements between subpopulations: (i) by sending a clone of an element from the source subpopulation to the target subpopulation (*pollination*); (ii) by moving an element from the source subpopulation to the target one (*plain migration*). If the subpopulations size should be kept constant, then in the pollination case for each new incorporated element, another element (e.g. a random one, or the worst one) is deleted. In the case of plain migration a replacing element (usually randomly selected) is sent from the target subpopulation to the source one.

In order to describe a random pollination process between s subpopulations by using communication rules specific to a membrane system, we consider the membrane structure described in Figure 1(a). Each elementary membrane corresponds to a subpopulation, and besides the objects corresponding to the elements in the subpopulation, it also contain some objects which are used for communication. These objects, denoted by r_{id}, are identifiers of the regions with which the subpopulation corresponding to the current region can communicate (in a fully connected topology of s subpopulations the identifiers belong to $\{1, \ldots, s\}$). On the other hand, when the migration step is initiated, a given set of copies of a migration symbol μ is created into each elementary membrane. The multiplicity of μ is related with the migration probability p_m (e.g. it is $\lfloor mp_m \rfloor$, where m is the size of subpopulation in the current region). Possible communication rules, for each type of membrane, describing the pollination process are presented in the following:

Elementary membranes. Let us consider the membrane corresponding to a subpopulation S_i. There are two types of rules: an *exporting* rule ensuring the transfer of an element to the skin membrane which plays the role of an communication environment, and an *assimilation* rule ensuring, if it is necessary, that the subpopulation size is kept constant.

An export rule can be described as:

$$R^{S_i}_{\text{export}} : \mu x^{S_i} r^{S_i}_{id} \rightarrow (x^{S_i}, here)(x^{S_i} r^{S_i}_{id} d, out) \tag{4}$$

An assimilation rule can be described as:

$$R^{S_i}_{\text{ass}} : dx^{S_i} \rightarrow \lambda \tag{5}$$

x^{S_i} denotes in both rules an arbitrary element from the subpopulation S_i, and $r^{S_i}_{id}$ identifies the region where clones of the elements from the subpopulation S_i can be sent. At each application of $R^{S_i}_{\text{export}}$ a copy of the symbol μ is consumed, and a copy of a deletion symbol d is created in the skin membrane.
Skin membrane. The communication rule corresponding to the skin membrane is:

$$R^0 : dx^{S_i} r^{S_i}_{id} \rightarrow (dx^{S_i}, in_{id}) \tag{6}$$

In the case of plain random migration, any element x^{S_i} from a source subpopulation S_i can be exchanged with an element x^{S_j} from a target subpopulation S_j. Such a communication process is similar with that in tissue P systems [7] described as $(i, x^{S_i}/x^{S_j}, j)$. Other communication policies (e.g. those based on elitist selection or replacement) can be similarly described.

4 A Hybrid Approach for Continuous Optimization

Based on the previous comparative analysis of membrane systems and distributed evolutionary algorithms we develop a new evolutionary algorithm which combines ideas from both membrane and distributed evolutionary systems.

Algorithm description. Let us consider a membrane structure consisting of a skin membrane containing s elementary membranes. Each elementary membrane i contains a subpopulation S_i on which a fully asynchronous evolutionary algorithm as that described in Section 2 is applied. Initially all subpopulations have the same size $m(0)$, but during the evolution their sizes can vary. The skin membrane contains also a subpopulation of elements, but different transformation rules are applied here (e.g. local search rules instead of evolutionary operators). This structure corresponds to a star communication topology (Figure 1b) which is less frequently used in distributed evolutionary algorithms. The communication is only between S_0 (corresponding to skin membrane) and the other subpopulations. The algorithm consists of two stages which are repeatedly applied until a stopping condition is satisfied. The general structure is described in Algorithm 4.

The *evolutionary stage* consists in applying an evolutionary algorithm on each of the subpopulations in inner membranes for τ iterations. The evolutionary stage is applied in parallel to all subpopulations. The subpopulations in inner membranes are initialized only at the beginning, thus the next evolutionary stage starts from the current state of the population. In this stage the only

transformation of the population in the skin membrane consists in applying a local search procedure to the best element of the population.

The *communication stage* consists in transferring clones of the best element from the inner membranes to the skin membrane by applying the rule $x_* \rightarrow (x_*, here)(x_*, out)$ in each elementary membrane. Moreover, the worst elements from the inner membranes are replaced with randomly selected elements from the skin membrane. If the subpopulation S_0 should have more than $s+1$ elements, then at each communication stage some randomly generated elements are added. The effect of such a communication strategy is that the worst elements in inner membranes are replaced with the best elements from other membranes or with randomly generated elements. In order to ensure the elitist character of the algorithm, the best element from the skin membrane is conserved at each step. It represents the approximation of the optimum we are looking for.

Algorithm 4 A Hybrid Approach

for all $i \in \{0, \ldots, s\}$ **do**
  Random initialization of the subpopulation S_i
end for
repeat
  for all $i \in \{1, \ldots, s\}$ **do**
    Apply an EA to S_i for τ steps
  end for
  Apply local search to the best element in S_0
  Reset subpopulation S_0 (all elements in S_0 excepting for the best one are deleted)
  for all $i \in \{1, \ldots, s\}$ **do**
    send a clone of the best element from S_i to S_0
  end for
  add random elements to S_0 (if its size should be larger than $s+1$)
  for all $i \in \{1, \ldots, s\}$ **do**
    Replace the worst element of S_i with a copy of a randomly selected element from S_0
  end for
until a stopping condition is satisfied

The evolutionary algorithm applied in each subpopulation S_i ($i = \overline{1,s}$) can be of any type. In our experimental analysis we used two variants:

Variant 1. The first variant is a generational algorithm which uses only one variation operator inspired from differential evolution algorithms [11]. It combines the recombination and mutation operators, so an offspring $z_i = \mathcal{R}(x_i, x_*, x_{r_1}, x_{r_2}, x_{r_3})$ is obtained by

$$z_i^j = \begin{cases} \gamma x_*^j + (1-\gamma)(x_{r_1}^j - x_*^j) + F(x_{r_2}^j - x_{r_3}^j)N(0,1), & \text{with probability } p \\ x_i^j, & \text{with probability} 1-p, \end{cases} \tag{7}$$

where r_1, r_2 and r_3 are random values from $\{1, \ldots, m\}$, x_* is the best element of the population, $F \in (0, 2]$, $p \in (0, 1]$ and $\gamma \in [0, 1]$. An entire population of offsprings $z_1 \ldots z_m$ is constructed by applying the above rule. The survivors are selected by comparing the parent x_i with its offspring z_i and by choosing the best one.

Variant 2. The second variant is based on the same recombination operator as in Variant 1, but it is combined with the fully asynchronous strategy described in Algorithm 3. The selection is based only on the deletion rule (Rule 3a). Each of these two types of rules (recombination combined with mutation and selection) is applied with a given probability (e.g. $p_R = p_S = 0.5$, $p_M = 0$).

Algorithm 4 is somewhat similar to the membrane algorithm proposed by Nishida in [6]. Both are hybrid approaches which combine evolutionary search with local search, and are based on a communication structure inspired by membrane systems. However there are some significant differences between these two approaches:

(i) they use different communication topologies: linear topology in the membrane algorithm of [6] vs. star topology in Algorithm 4; therefore they use different membrane structures (see Figure 2);
(ii) they address different classes of optimization problems: combinatorial optimization vs. continuous optimization;
(iii) they are based on different evolutionary rules (genetic crossover and mutation in [6] vs. differential evolution recombination here), and different local search procedures (tabu search in [6] vs. Nelder-Mead local search [9] in the current approach);
(iv) they are characterized by different granularity: micro-populations (e.g. two elements) but a medium number of membranes (e.g. 50) in [6] vs. medium sized subpopulations (e.g. 10) but a small number of membranes (e.g. 5);
(v) they are characterized by different communication frequencies: transfer of elements between membranes at each step in the membrane algorithm vs. transfer of elements only after τ evolutionary steps have been executed (e.g. $\tau = 100$).

Experimental analysis. In order to analyze the ability of the fully asynchronous strategy (Algorithm 3) and of the hybrid approach described above (Algorithm 4) to approximate the solution of continuous optimization problems, we conducted some preliminary numerical experiments. The algorithms have been applied to some classical test functions (see Table 1) used in empirical analysis of evolutionary algorithms. All these problems are of minimization type, and the optimal value is 0. In all these tests the problem size was $n = 30$. The domains are $[-100, 100]^n$ for sphere function, $[-32, 32]^n$ for Ackley's function and $[-600, 600]^n$ for Rastrigin's function.

The first set of experiments aimed to compare the classical generational and steady state strategies for panmictic EAs (Algorithms 1 and 2) with the non-standard one inspired from rules application in membrane systems (Algorithm 3). In all cases the recombination rule was that of differential evolution type

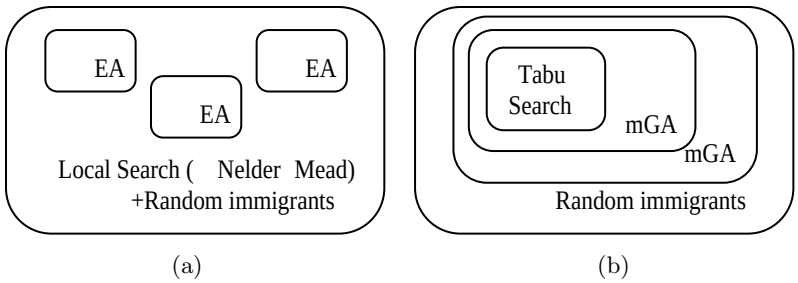

Fig. 2. (a) Membrane structure of Algorithm 4.(b) Membrane structure of Nishida's approach.

given by Equation (7). The parameters controlling the evolutionary algorithm are chosen as follows: $m = 50$ (population size), $p = F = 0.5$ (the control parameters involved in the recombination rule given in Equation (7)), $f_* = 10^{-5}$ (accuracy of the optimum approximation).

Table 1. Test functions

Name	Expression
Sphere	$f(x) = \sum_{i=1}^{n} x_i^2$
Ackley	$f(x) = -20\exp\left(-0.2\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$
Griewank	$f(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(x_i/\sqrt{i}) + 1$

We consider that the search process is successful whenever it finds a configuration for which the objective function has a value less than f_*. The ratio of successful runs from a set of independent runs (in our tests the number of independent runs of the same algorithm for different randomly initialized populations was 30) is a measure of the effectiveness of the algorithm. As a measure of efficiency we use the number nfe of objective function evaluations, both average value and standard deviation.

Table 2 presents comparative results for generational, steady state and fully asynchronous strategies for two variants ($\gamma = 0$ and $\gamma = 1$) of the evolutionary operator described by Equation (7). The results of Table 2 suggest that the fully asynchronous strategy (Algorithm 3) behaves differently than classical generational and steady state strategies: it is worse in the case of recombination rules characterized by $\gamma = 0$, and better in the case of recombination rules characterized by $\gamma = 1$. This can be explained by the fact that applying the evolutionary rules in an asynchronous manner reduces the selection pressure. By combining such a strategy with evolutionary rules characterized by strong

exploitative character (as is Equation(7) with $\gamma = 1$) allows avoiding premature convergence, while by combining it with a more explorative rule (as in the case of $\gamma = 0$) lead to a slower convergence.

Table 2. Comparison of evolutionary rules applying strategies in a panmictic EA.

Test function	Generational Success	Generational $\langle nfe \rangle$	Steady state Success	Steady state $\langle nfe \rangle$	Fully asynchronous Success	Fully asynchronous $\langle nfe \rangle$
			$\gamma = 0$			
Sphere	30/30	27173±456	30/30	25031±505	30/30	47286± 5170
Ackley	30/30	37490±653	30/30	34785±495	30/30	65364± 6239
Griewank	30/30	28885±846	30/30	26816±1026	22/30	49430 ± 6894
			$\gamma = 1$			
Sphere	0/30	-	30/30	9515±527	30/30	6525± 2274
Ackley	13/30	15876±2015	0/30	-	30/30	7968± 2798
Griewank	7/30	11885±1255	4/30	8837±439	22/30	9586 ± 5229

The second set of experiments aimed to analyze the approximation ability of the hybrid Algorithm 4. Two variants of this algorithm (one based on a generational evolutionary algorithm, and the other based on the fully asynchronous variant) are compared with a more classical generational differential evolution combined with a random communication strategy [14]. All variants use the recombination rule described by Equation (7) for $\gamma = 1$. The other parameters have been chosen as in the first set of experiments. The parameters specific to distributed evolutionary algorithms have been chosen as follows: $m(0) = 10$ (initial size of subpopulations; when Algorithm 4 is applied, this size varies between $m(0)/2 = 5$ and $2m(0) = 20$), $s = 5$ (the number of subpopulations/membranes), $\tau = 100$ (the number of evolutionary steps between two communication steps). In order to reduce the computational cost, the local search (Nelder Mead procedure) is not activated at each communication step, but only after 10 communication steps.

Table 3. Behaviour of distributed EAs: generational EA with random migration vs. the hybrid approach

Test function	Generational and random migration Success	Generational and random migration $\langle nfe \rangle$	Algorithm 4 (variant 1) Success	Algorithm 4 (variant 1) $\langle nfe \rangle$	Algorithm 4 (variant 2) Success	Algorithm 4 (variant 2) $\langle nfe \rangle$
Sphere	30/30	62002±5123	30/30	59049±527	30/30	3771± 722
Ackley	1/30	84970	30/30	240675±55217	30/30	3173± 880
Griewank	20/30	62902±4272	12/30	126304±89874	26/30	48724 ± 5044

The results of Table 3 show that the communication strategy based on the membrane structure described in Figure 2(a) combined with the fully asynchronous evolutionary strategy analyzed in the previous experiment is effective and efficient when is compared to the other two strategies.

These results suggest that structuring the population as in membrane systems, and applying the evolutionary operators in an unordered manner, then we obtain evolutionary algorithms with a new dynamics. This new dynamics leads to significantly better results for certain evolutionary operators and test functions (see results in Table 3 for variant 2 of Algorithm 4). However the hybrid approach is not superior to the classical generational variant combined with a random migration for some evolutionary operators (e.g. variant 1 of Algorithm 4). Such a situation is not unusual in evolutionary computing, being accepted that no evolutionary algorithm is superior to all the others with respect to all problems [13].

5 Conclusions

As it has been recently stated in [8], the membrane community is looking for a relationship, a link between membrane systems and distributed evolutionary algorithms. We claim that the main similarity is at a conceptual level, and each important concept in distributed evolutionary computing has a correspondent in membrane computing. This correspondence is summarized in the following table:

Membrane system	Distributed Evolutionary Algorithm
Membrane(region)	Population
Objects	Individuals
Evolution rules	Evolutionary operators
Membrane structure	Communication topology
Communication rules	Communication policy

Besides these conceptual similarities, there are some important differences:

(i) membrane systems have an exact notion of computation, while evolutionary computation is an approximate one;
(ii) membrane computing is based on symbolic representations, while evolutionary computing is mainly used together with numerical representations.

Despite these differences, ideas from membrane computing are useful in developing new distributed meta-heuristics. A first attempt was given by the membrane algorithm proposed in [6]. However this first approach did not emphasized at all the important similarities between membrane computing and distributed evolutionary computing. This aspect motivates us to start a depth analysis of these similarities, having the aim of describing the evolutionary algorithms by using the formalism of membrane computing. As a result of this analysis, we present in this paper a non-standard strategy of applying the evolutionary operators. This strategy, characterized by an arbitrary application of evolutionary operators, proved to be better than the classical generational and steady state strategies when applied for some continuous optimization problems. On the other hand, based on the relationship between membrane structures and communication topologies, we introduce a new hybrid distributed evolutionary algorithm effective in

solving continuous optimization problems. There are important and significant differences between our approach and that proposed in [6], and we presented them in Section 4. One important difference between these two approaches is related to the efficiency of a parallel implementation, which is influenced by the communications costs [14]. In our approach the communication rules are applied less frequently than in the membrane algorithm, and this leads to a lower communication cost and a more efficient parallel implementation. Algorithms 3 and 4 proposed and analyzed in this paper are good and reliable in approximating solutions of optimization problems. This fact proves that by using ideas from membrane computing, new distributed metaheuristic methods can be developed.

References

1. E. Alba, M. Tomassini. Parallelism and Evolutionary Algorithms, *IEEE Transactions on Evolutionary Computation*, **6**(5), pp.443-462, 2002.
2. G. Ciobanu. Distributed Algorithms over Communicating Membrane Systems, *BioSystems* **70**(2), pp.123-133, 2003.
3. A.E. Eiben, J.E. Smith. *Introduction to Evolutionary Computing*, Springer, 2002.
4. F. Herrera, M. Lozano. Gradual Distributed Real-Coded Genetic Algorithms, *IEEE Transactions on Evolutionary Computation*, **4**1, pp.43-63, 2002.
5. J.J. Hu, E. D. Goodman. The Hierarchical Fair Competition (HFC) Model for Parallel Evolutionary Algorithms, *Proceedings of Congress of Evolutionary Computation*, IEEE Computer Society Press, pp. 49-54, 2002.
6. T.Y. Nishida. An application of P system: A new algorithm for NP-complete optimization problems, in eds. N. Callaos, et. al., *Proceedings of the 8th World Multi-Conference on Systems, Cybernetics and Informatics* , **V**, pp. 109-112, 2004.
7. Gh. Păun. *Membrane Computing. An Introduction*, Springer, 2002.
8. Gh. Păun. Further Twenty-Six Open Problems in Membrane Computing, *Third Brainstorming Meeting on Membrane Computing* (online document, http://psystems.disco.unimib.it), 2005.
9. W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling. *Numerical Recipes in C.*, Cambridge University Press, 2002.
10. G. Rudolph. Convergence of Evolutionary Algorithms in General Search Spaces, in *Proc. of the third Congress on Evolutionary Computation*, IEEE Computer Society Press, pp. 50-54, 1996.
11. R. Storn, K. Price. Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Technical Report TR-95-012*, ICSI, 1995.
12. M. Tomassini. Parallel and Distributed Evolutionary Algorithms: A Review, in K. Miettinen, M. Mäkelä, P. Neittaanmki and J. Periaux (eds.): *Evolutionary Algorithms in Engineering and Computer Science*, J. Wiley and Sons, pp.113-133, 1999.
13. D.H. Wolpert, W.G. Macready. No Free Lunch Theorems for Optimization, *IEEE Transactions on Evolutionary Computing*, **1**, pp. 67-82, 1997.
14. D. Zaharie, D. Petcu. Parallel Implementation of Multi-population Differential Evolution, in D. Grigoras, A. Nicolau (eds), *Concurrent Information Processing and Computing*, IOS Press, pp. 223-232, 2005.