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 an analysis of the similarities between distributed evolutionary algorithms and membrane systems. The correspondences between evolutionary operators and evolution rules and between communication topologies and policies in distributed evolutionary algorithms and membrane structures and communication rules in membrane systems are identified. As a result of this analysis we propose new strategies of applying the operators in evolutionary algorithms and new variants of distributed evolutionary algorithms. The behavior of these variants is numerically tested for some 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 [13].

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. Because of this one-to-one correspondence we sometimes use membrane instead of region. The space outside the skin membrane is called the environment. 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

H.J. Hoogeboom et al. (Eds.): WMC 7, LNCS 4361, pp. 536–553, 2006.
© Springer-Verlag Berlin Heidelberg 2006

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 [8].

Evolutionary algorithms are reliable methods 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?*

A first attempt to build a bridge between membrane computing and evolutionary computing is given by T.Y Nishida, in [7], 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 propose new strategies for applying evolutionary rules and new variants of distributed evolutionary algorithms inspired by membrane systems structure and functioning.

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, more flexible, 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 distributed evolutionary algorithms on the other hand. As a result of this analysis is proposed a new variant of distributed evolutionary algorithms. Section 4 is devoted to a numerical analysis of the membrane systems inspired variants of evolutionary algorithms.

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 recombination 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. This can be modelled by a mapping $\mathcal{M} : D \to D$ defined by $\mathcal{M}((u^1, \dots, u^n)) = (v^1, \dots, v^n)$. Typical examples are:

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

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 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^q to D^r and can be used in two main situations: selection of parents for recombination (in this case $q = m$, $r < m$, and the parents selection is not necessarily based on the quality of elements), and selection of survivors (in this case $q \geq m$, $r = m$, 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 population of new elements or to the joined population of parents and offsprings. The general structure of a generational EA is presented in Algorithm 1 where $X(t)$ denotes the population corresponding to generation t, z denotes an offspring 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:    Z := ∅
5:    for all i ∈ {1, ..., m} do
6:       Z := Z ∪+ (M̄ ∘ R ∘ S_p)(X(t))
7:    end for
8:    X(t + 1) := S_s(X(t) ∪+ Z)
9:    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.

The simplest way to interpret 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

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

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 of population elements. 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 initial size of the population), then the operators $\mathcal{R}$ and $\mathcal{M}$ are inhibited.

The role of selection is to modify the distribution of elements in the population by eliminating or by cloning some elements. Simple selection operators 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 (e.g., half of the initial size of the population).

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.

Rule 4 (insertion of random elements): $x \to x\xi$, where $\xi \in D$ is a randomly generated element and x is an arbitrary element of the population.

The last rule does not correspond to the classical evolutionary operators but is used in evolutionary algorithms in order to stimulate the population diversity.

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, based on a probabilistic decision concerning the operator to be applied at each step, is described in Algorithm 3. The rules involved in the evolutionary process are: recombination, mutation, selection by deletion, selection by cloning and random elements insertion. The probabilities corresponding to these rules are p_R, p_M, p_{Sd}, p_{Sc} and $p_I \in [0, 1]$. By applying the evolutionary operators in such a probabilistic way, we obtain a flexible algorithm which works with variable size populations. In Algorithm 3 the population size corresponding to iteration t is denoted by $m(t)$. Even if variable, the population size is limited by a lower bound, m_*, and an upper bound, m^*.

Algorithm 3. Evolutionary algorithm with random selection of operators

Random initialization of 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 $(u < p_R) \wedge (m(t) < m^*)$ **then**
    apply *Rule 1 (recombination)*
  end if
  if $(u \in [p_R, p_R + p_M)) \wedge (m(t) < m^*)$ **then**
    apply *Rule 2 (mutation)*
  end if
  if $(u \in [p_R + p_M, p_R + p_M + p_{Sd})) \wedge (m(t) > m_*)$ **then**
    apply *Rule 3a (selection by deletion)*
  end if
  if $(u \in [p_R + p_M + p_{Sd}, p_R + p_M + p_{Sd} + p_{Sc})) \wedge (m(t) < m^*)$ **then**
    apply *Rule 3b (selection by cloning)*
  end if
  if $(u \in [p_R + p_M + p_{Sd} + p_{Sc}, 1]) \wedge (m(t) < m^*)$ **then**
    apply *Rule 4 (insertion of a random element)*
  end if
  $t := t + 1$
until a stopping condition is satisfied

An important feature of Algorithm 3 is given by the fact that only one operator is applied at each step, and thus it can be considered as an operator oriented approach. This means that first an operator is probabilistically selected and only afterwards are selected the elements on which it is applied.

Another approach would be that oriented toward elements, meaning that at each step all elements can be involved in a transformation and for each one is selected (also probabilistically) the rule to be applied. After such a parallel step, a mechanism of regulating the population size can be triggered. If the population became too small, then selection by cloning can be applied or some

random elements could be inserted. If the population became too large, then selection by deletion could be applied. This strategy is characterized through a parallel application of rules, thus it is more in the spirit of membrane computing. However, this strategy did not provide better results than Algorithm 3 when it was tested for continuous optimization problems.

We can expect that the behavior of such algorithms be different from the behavior of more classical generational and steady state algorithms. However, from a theoretical viewpoint, such an algorithm can be still modeled by a Markov chain and the convergence results still hold [11]. 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.

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 strategy with random selection of operators 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 [13]: (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 [13]. 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 behavior 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 modeled 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 neighbor 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 elements either in a inner or in the outer membrane (see Figure 1).

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

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 can contain also evolution rules.

Linear topology. In this case a subpopulation p can communicate only with its neighbor subpopulations $p+1$ and $p-1$. The corresponding structure is given

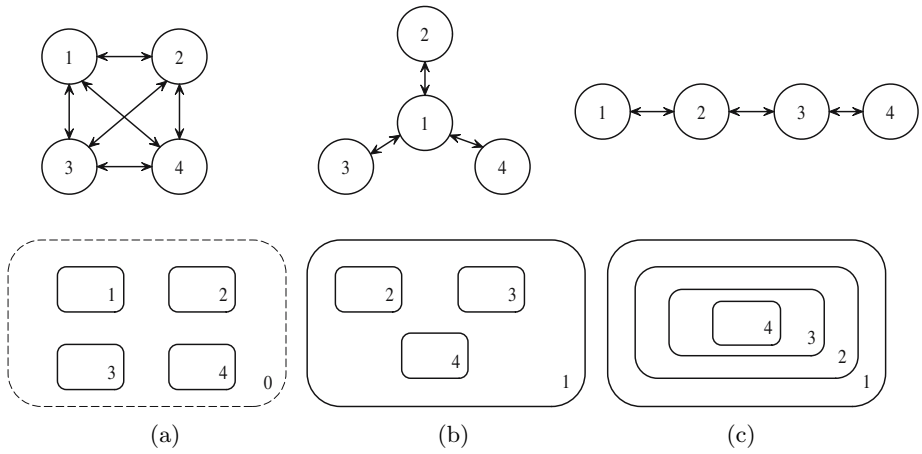

Fig. 1. Communication topologies in distributed evolutionary algorithms and their corresponding membranes structures. (a) Fully connected topology; (b) Star topology; (c) Linear topology.

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. The 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.

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*). An element is selected with a given probability, p_m, usually called migration probability. 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 number of copies of a migration symbol η are created into each elementary membrane. The multiplicity of η 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 ($i \neq 0$). 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.
The export rule can be described as:

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

The 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 η 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 [8] 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.

3.3 Distributed Evolutionary Algorithms Inspired by Membrane Systems

A first communication strategy inspired by membrane systems is that used in the membrane algorithm proposed in [7] and also in [6]. The membrane algorithm

proposed in [7] can be interpreted as a hybrid distributed evolutionary algorithm based on a linear topology (Figure 1c) and a tabu search heuristic. The basic idea of communication between membranes is that of moving the best element in the inner membrane and the worst one in the outer membrane. The skin membrane receives random elements from the environment. The general structure of such an algorithm in presented in Algorithm 4.

Algorithm 4. Distributed evolutionary algorithm based on a linear topology

for all $i \in \{1, \ldots, s\}$ **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_1
  for all $i \in \{1, \ldots, s-1\}$ **do**
    send a clone of the best element from S_i to S_{i+1}
  end for
  for all $i \in \{2, \ldots, s\}$ **do**
    send a clone of the worst element from S_i to S_{i-1}
  end for
  Add a random element to S_1
  for all $i \in \{1, \ldots, s\}$ **do**
    Delete the two worst elements of S_i
  end for
until a stopping condition is satisfied

Another communication topology, corresponding to a simple membrane structure but not very common in distributed evolutionary computing, is that of star type (Figure 1b). In the following we propose a hybrid distributed evolutionary algorithm based on this topology. Let us consider a membrane structure consisting of a skin membrane containing $s-1$ elementary membranes. Each elementary membrane i contains a subpopulation S_i on which an evolutionary algorithm is applied. This evolutionary algorithm can be based on a random application of rules. 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). The communication is only between S_1 (corresponding to skin membrane) and the other subpopulations. The algorithm consists of two stages: an evolutionary one and a communication one which are repeatedly applied until a stopping condition is satisfied. The general structure is described in Algorithm 5.

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 sending 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_1 should have more than s 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 5. Distributed evolutionary algorithm based on a star topology

1: **for all** $i \in \{1, \ldots, s\}$ **do**
2: Random initialization of the subpopulation S_i
3: **end for**
4: **repeat**
5: **for all** $i \in \{2, \ldots, s\}$ **do**
6: Apply an EA to S_i for τ steps
7: **end for**
8: Apply local search to the best element in S_1
9: Reset subpopulation S_1 (all elements in S_1 excepting for the best one are deleted)
10: **for all** $i \in \{2, \ldots, s\}$ **do**
11: send a clone of the best element from S_i to S_1
12: **end for**
13: add random elements to S_1 (if its size should be larger than s)
14: **for all** $i \in \{2, \ldots, s\}$ **do**
15: Replace the worst element of S_i with a copy of a randomly selected element from S_1
16: **end for**
17: **until** a stopping condition is satisfied

4 Numerical Results

The aim of the experimental analysis was twofold: (i) to compare the behavior of the evolutionary algorithms with random application of operators (Algorithm 3) and of those based on generational and steady-state strategies (Algorithms 1 and 2); (ii) to compare the behavior of distributed evolutionary algorithms based on linear and star topologies (Algorithm 4 and Algorithm 5) with that of an algorithm based on a fully connected topology and random migration [15].

The particularities of the evolutionary algorithm applied in each subpopulation and the values of the control parameters used in the numerical experiments are presented in the following.

Evolutionary operators. The generation of offsprings is based on only one variation operator inspired from differential evolution algorithms [12]. 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

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

where r_1, r_2 and r_3 are random values from $\{1, \ldots, m\}$, x_* is the best element of the population, $F \in (0, 2]$, $p \in (0, 1]$, $\gamma \in [0, 1]$ and $U(a_j, b_j)$ is a random value, uniformly generated in domain of values for component j.

In the generational strategy 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. In the sequential strategy, at each step is generated one offspring which replaces, if it is better, the worst element of the population.

In the variant based on Algorithm 3 the following rules are probabilistically applied: the recombination operator given by Equation (7) is applied with probability p_R, the selection by deletion is applied with probability p_{Sd} and the insertion of random elements is applied with probability p_I. Since there are two variants of the recombination operator (for $\gamma = 0$ and for $\gamma = 1$) each one can be applied with a given probability: p_R^0 and p_R^1. These probabilities satisfy $p_R^0 + p_R^1 = p_R$.

Test functions. 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, the optimal solution being $x^* \in D$ and the optimal value being $f^* \in [-1, 1]$. x^* and f^* have been randomly chosen for each test problem. In all these tests the problem size was $n = 30$. The domains values of decision variables are $D = [-100, 100]^n$ for sphere function, $D = [-32, 32]^n$ for Ackley's function, $D = [-600, 600]^n$ for Griewank's function and $D = [-5.12, 5.12]^n$ for Rastrigin's function.

Parameters of the algorithms. 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)), $\epsilon = 10^{-5}$ (accuracy of the optimum approximation).

We consider that the search process is successful whenever it finds a configuration, x_*, for which the objective function has a value which satisfies $|f^* - f(x_*)| < \epsilon$ by using less than 500000 objective functions evaluations. 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 1. Test functions

Name	Expression
Sphere	$f_1(x) = f^* + \sum_{i=1}^{n}(x_i - x_i^*)^2$
Ackley	$f_2(x) = f^* - 20\exp\left(-0.2\sqrt{\frac{\sum_{i=1}^{n}(x_i - x_i^*)^2}{n}}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi(x_i - x_i^*))\right) + 20 + e$
Griewank	$f_3(x) = f^* + \frac{1}{4000}\sum_{i=1}^{n}(x_i - x_i^*)^2 - \prod_{i=1}^{n}\cos((x_i - x_i^*)/\sqrt{i}) + 1$
Rastrigin	$f_4(x) = f^* + \sum_{i=1}^{n}((x_i - x_i^*)^2 - 10\cos(2\pi(x_i - x_i^*))) + 10$

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

Test	Generational		Steady state		Algorithm 3	
fct	Success	$\langle nfe\rangle$	Success	$\langle nfe\rangle$	Success	$\langle nfe\rangle$
f_1	30/30	27308±396	30/30	25223±469	30/30	24758± 1328
f_2	30/30	37803±574	30/30	35223±611	29/30	27461± 2241
f_3	30/30	29198±1588	30/30	27010±1016	28/30	20017 ± 1640
f_4	19/30	335518±55107	18/30	296111±49316	29/30	193741± 113126

Results. Table 2 presents comparative results for generational, steady state and the strategy based on random selection of operators (Algorithm 3). For the first two strategies the evolutionary operator described by Equation (7) was applied for $\gamma = 0$. For $\gamma = 1$ the success ratio of generational and sequential variants is much smaller, therefore these results are not presented. The probabilities for applying the evolutionary operators in Algorithm 3 were $p_R = 0.5$ ($p_R^0 = 0.35$,$p_R^1 = 0.15$), $p_{Sd} = 0.5$, $p_I = 0$. The initial population size was $m(0) = 50$ and the lower and upper bounds were $m_* = m(0)/2$ and $m^* = 2m(0)$ respectively.

The results of Table 2 suggest that for the functions f_1, f_2, f_3 the Algorithm 3 does not prove to be superior to generational and steady state strategies. However a significant improvement can be observed for function f_4 which is a difficult problem for EA based on recombination as in Equation (7). However, by changing the probability p involved in the recombination operator (e.g., $p = 0.2$ instead of $p = 0.5$) a good behavior can be obtained also by generational and steady state strategies. On the other hand, by dynamically adjusting the probabilities of applying the evolutionary operators the behavior of Algorithm 3 can be improved. For instance, if one choose $p_R = 0$, $p_{Sd} = 0.1$ and $p_I = 0.9$ whenever the average variance of the population is lower than 10^{-8}, then in the case of Ackley function the success ratio is 30/30 and $\langle nfe\rangle$ is 19312 with a standard deviation of 13862.

The second set of experiments aimed to compare the communication strategies inspired by membranes (Algorithm 4 and Algorithm 5) with a communication strategy characterized by a fully connected topology and a random migration of elements [15]. In all experiments the number of subpopulations was $s = 5$, the initial size of each subpopulation was $m(0) = 10$ and the number of evolutionary steps between two communication stages was $\tau = 100$. In the case of random migration the probability of selecting an element for migration was $p_m = 0.1$.

Table 3. Behavior of distributed EAs based on a generational EA

Test fct.	Fully connected topology and random migration		Linear topology (Algorithm 4)		Star topology (Algorithm 5)	
	Success	$\langle nfe \rangle$	Success	$\langle nfe \rangle$	Success	$\langle nfe \rangle$
f_1	30/30	30500±1290	30/30	30678±897	30/30	34943± 4578
f_2	30/30	41000±2362	30/30	41349±1864	30/30	51230± 8485
f_3	20/30	32500±2449	30/30	33013±3355	26/30	41628 ± 8353
f_4	7/30	158357±66647	4/30	95538±10610	24/30	225280 ± 132636

The results in Table 3 show that the communication strategy based on the linear topology (Algorithm 4) behaves almost similarly to the strategy based on fully connected topology and random migration. On the other hand, the communication strategy based on the star topology (Algorithm 5) has a different behavior, characterized by a slower convergence. This behavior can be explained by the higher degree of randomness induced by inserting random elements in the skin membrane. However this behavior can be beneficial in the case of difficult problems (e.g., Rastrigin) by avoiding premature convergence situations. In the case of Rastrigin's function the success ratio of Algorithm 5 is significantly higher than in the case of the other two variants.

Table 4. Behavior of distributed EAs based on the random application of evolutionary operators

Test fct	Fully connected topology and random migration		Linear topology (Algorithm 4)		Star topology (Algorithm 5)	
	Success	$\langle nfe \rangle$	Success	$\langle nfe \rangle$	Success	$\langle nfe \rangle$
f_1	30/30	42033±4101	30/30	59840±11089	30/30	98280± 20564
f_2	30/30	117033±80778	30/30	156363±103488	29/30	266783± 96539
f_3	15/30	51065±14151	17/30	72901±38654	21/30	111227 ± 58491
f_4	30/30	94822±22487	30/30	107412±25363	30/30	111752 ± 19783

The results in Table 4 show that by using the Algorithm 3 in each subpopulation the convergence is significantly slower for Ackley and Griewank functions while it is significantly improved in the case of Rastrigin function, both with respect to other distributed variants and with the panmictic algorithms used in the experimental analysis.

These results suggest that structuring the population as in membrane systems, and applying the evolutionary operators in an unordered manner, 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 4 for Rastrigin's function). However the hybrid approach is not superior to the classical generational variant combined with a random migration for the other test functions. 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 [14].

Algorithm 5 is somewhat similar to the membrane algorithm proposed by Nishida in [7]. 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 [7] vs. star topology in Algorithm 5; 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 [7] vs. differential evolution recombination here), and different local search procedures (tabu search in [7] vs. Nelder-Mead local search [10] 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 [7] 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., $\tau = 100$).

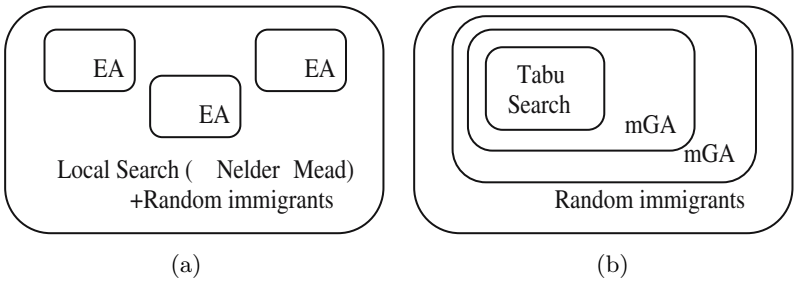

Fig. 2. (a) Membrane structure of Algorithm 5.(b) Membrane structure of Nishida's approach.

5 Conclusions

As it has been recently stated in [9], 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 [7]. 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 behave differently 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 some continuous optimization problems. The algorithms 3 and 5 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.

Besides this way of combining membrane and evolutionary computing there are at least two other research directions which deserve further investigation:

(i) the use evolutionary algorithms to evolve membrane structures;
(ii) the use of membrane systems formalism in order to understand the behavior of distributed evolutionary algorithms.

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. A. Leporati, D. Pagani. A Membrane Algorithm for the Min Storage Problem in H.J. Hoogeboom, Gh. Păun, G. Rozenberg (eds.), *Pre-Proceedings of the 7th Workshop on Membrane Computing*, 17-21 July 2006, Leiden, pp. 397–416, 2006.
7. T.Y. Nishida. An Application of P Systems: A New Algorithm for NP-complete Optimization Problems, in N. Callaos, et al. (eds.), *Proceedings of the 8th World Multi-Conference on Systems, Cybernetics and Informatics*, **V**, pp. 109–112, 2004.
8. Gh. Păun. *Membrane Computing. An Introduction*, Springer, 2002.
9. Gh. Păun. Further Twenty-Six Open Problems in Membrane Computing, *Third Brainstorming Meeting on Membrane Computing* (online document, `http://psystems.disco.unimib.it`), 2005.
10. W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling. *Numerical Recipes in C*, Cambridge University Press, 2002.
11. 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.
12. R. Storn, K. Price. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Technical Report TR-95-012*, ICSI, 1995.
13. 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.
14. D.H. Wolpert, W.G. Macready. No Free Lunch Theorems for Optimization, *IEEE Transactions on Evolutionary Computing*, **1**, pp. 67–82, 1997.
15. 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.