Int. J. Appl. Math. Comput. Sci., 2004, Vol. 14, No. 3, 317–333

ADVANCES IN PARALLEL HETEROGENEOUS GENETIC ALGORITHMS FOR CONTINUOUS OPTIMIZATION

ENRIQUE ALBA*, FRANCISCO LUNA*, ANTONIO J. NEBRO*

* Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga. E.T.S.I. Informática
Campus de Teatinos, 29071 Málaga, Spain
e-mail: {eat, flv, antonio}@lcc.uma.es

In this paper we address an extension of a very efficient genetic algorithm (GA) known as *Hy3*, a physical parallelization of the gradual distributed real-coded GA (GD-RCGA). This search model relies on a set of eight subpopulations residing in a cube topology having two faces for promoting exploration and exploitation. The resulting technique has been shown to yield very accurate results in continuous optimization by using crossover operators tuned to explore and exploit the solutions inside each subpopulation. We introduce here a further extension of *Hy3*, called *Hy4*, that uses 16 islands arranged in a hypercube of four dimensions. Thus, two new faces with different exploration/exploitation search capabilities are added to the search performed by *Hy3*. We analyze the importance of running a synchronous versus an asynchronous version of the models considered. The results indicate that the proposed *Hy4* model overcomes the *Hy3* performance because of its improved balance between exploration and exploitation that enhances the search. Finally, we also show that the async *Hy4* model scales better than the sync one.

Keywords: parallel genetic algorithms, continuous optimization, premature convergence, heterogeneity

1. Introduction

Evolutionary algorithms (EAs) are stochastic search methods that have been successfully applied in many search, optimization, and machine learning problems (Bäck *et al.*, 1997; Goldberg, 1989; Holland, 1975; Michalewicz, 1992). Unlike most other optimization techniques, EAs maintain a population of tentative solutions that are competitively manipulated by applying some variation operators to find a satisfactory, if not globally, optimum solution. Among the well-accepted subclasses of EAs (Bäck, 1996), genetic algorithms (GAs) (Goldberg, 1989; Holland, 1975) have been widely studied. The goal of this paper is to extend a previous work (Alba *et al.*, 2003; 2004) dealing with a new model for optimization in continuous domains with GAs.

GAs iteratively improve a population of individuals by applying a recombination operator (merging two or more parents to yield one or more offsprings) and a mutation of their contents (random alterations of the problem variables). However, if we stick to natural evolution, we should not operate on a single population in which a given individual has the potential to mate with any other partner in the same population (panmixia). Instead, species evolve in structured neighborhoods, and tend to reproduce within subgroups. Among the existing types of structured GAs, distributed GAs (dGAs) (Alba and Troya, 1999) are especially popular. Distributed evolutionary algorithms are a subclass of decentralized evolutionary algorithms (Alba and Tomassini, 2002) aimed at reducing the probability of convergence to local optima, promoting diversity, and finding alternative solutions to the same problem. Their advantage lies in partitioning the population into several subpopulations, each being processed by a GA, independently of the others. Furthermore, a sparse migration of individuals produces an exchange of genetic material among the subpopulations that usually improves the accuracy and efficiency of the algorithm.

By making different decisions on the component subalgorithms in a dGA, we obtain the so-called *heterogeneous* dGAs (multi-resolution methods). One way of constructing a heterogeneous dGA is through the application of different search strategies in each component algorithm. This means that the search occurs at multiple exploration and exploitation levels at the same time. In this paper we extend a heterogeneous dGA called *Hy3* (Alba *et al.*, 2003; 2004). *Hy3* is, in turn, a parallel extension of another heterogenous dGA named *gradual distributed real-coded GA* (GD-RCGA) (Herrera and Lozano, 2000). This model of search is an example of the distributed technique that runs eight populations concurrently in a cubic topology with sparse migrations of individuals among them. The GD-RCGA model is suitable for the optimization of continuous functions, because it includes in the basic improvement loop of the algorithm the utilization of crossover operators specialized for float genes (variables), engineered with fuzzy logic technology to deal explicitly with the traditional "fuzzy" GA concepts of exploration and exploitation.

There exist some studies on GD-RCGA in the literature (Herrera and Lozano, 2000). However, although the algorithm offers a straightforward parallelization, only sequential implementations exist. In them, a concurrent execution of the islands is simulated at hand on a monoprocessor. The *Hy3* algorithm presented in (Alba *et al.*, 2003; 2004) provides the first parallel implementation that actually runs in a cluster of machines. The contribution of the present work is, first, to propose a further extension of the *Hy3* model, called *Hy4* (*Hypercube4*). This new model is configured as a hypercube of four dimensions with 16 subpopulations, where a more advantageous balance between exploration/exploitation could be achieved. Additionally, we are interested in investigating the advantages that could outcome from an asynchronous design, instead of the synchronous search that the basic GD-RCGA suggests. Thus, *Hy4* comes as a new parallel model in which new numerical and efficiency challenges need to be studied.

The paper is organized as follows. We first outline a taxonomy of heterogeneous dGAs in the next section. Section 3 presents the background to understand the *Hy4* model and a discussion on our parallel implementations. In Section 4, we briefly introduce the problems contained in our benchmark. In Section 5, we present the parameterization that we used here. In the next section, we analyze the results from a numerical and run time point of view. Finally, we summarize the conclusions and discuss several lines for future research in Section 7.

2. Heterogeneous Distributed Genetic Algorithms

One of the main difficulty in heuristics is premature convergence (e.g., in GAs (Baker, 1987; de Jong, 1975)). It arises when the search is likely to be trapped in a region that does not contain the global optimum. An approach to address this problem focuses on keeping the diversity of the population high. The lack of diversity in the population may be provoked in turn by the loss of critical alleles due to selection, the disruption due to crossover, or a poor parameter setting (Herrera and Lozano, 2000; Potts *et al.*, 1994), among other things.

In this sense, diversity preservation methods based on *spatial separation* have been proposed in order to avoid premature convergence (Manderick and Spiessens, 1989; Mühlenbein *et al.*, 1991; Tanese, 1989). One of the most important examples of such a kind of algorithms are distributed GAs (Alba and Troya, 1999). In dGAs, an attempt to overcome the premature convergence problem is made by preserving diversity due to the semi-isolation of the subpopulations.

Distributed GAs may be classified into the following two categories, with respect to subpopulation homogeneity (Lin *et al.*, 1994):

- *Homogeneous dGAs*. Every subpopulation performs the same kind of search (same genotype, operators, etc.) on different sets of randomly generated individuals. They are considered as a direct extension of the canonical GA, and most dGAs proposed in the literature are members of this category (Mühlenbein *et al.*, 1991; Tanese, 1989).

- *Heterogeneous dGAs*. The subpopulations are processed using GAs with either different control parameter values, or genetic operators, or an encoding scheme, etc. The result is a robust multi-resolution search method that can be explicitly tuned to carry out exploration and exploitation depending on the problem. A taxonomy of this sort of dGAs is presented in this section.

The homo/heterogeneity could be understood as a term referring to the execution platform, where each island executes over a different hardware or operating system (Alba *et al.*, 2002). However, there exist different levels for heterogeneity as regards the kind of search that the islands make. At this "software" or numeric level, we can also distinguish various sublevels according to the source of heterogeneity:

1. *Parameter level*. The first approach to achieve the numeric heterogeneity is to use the same GA in each island with different parameters of selection, recombination, mutation, and/or migration. These parameters could be initially preprogrammed (Adamidis and Petridis, 1996; 2002), randomly chosen during the evolution (Hiroyasu *et al.*, 1999; Miki *et al.*, 1999), or they could follow an adaptive strategy (Hinterding *et al.*, 1996; Schlierkamp-Voosen and Mühlenbein, 1994; Schnecke and Vornberger, 1996).

2. *Operator level*. At this level, heterogeneity is introduced by using different genetic operators into the same GAs (Herrera and Lozano, 1997; Herrera *et al.*, 1998).

3. *Representation level*. This is a more subtle kind of heterogeneity, where each subpopulation stores locally encoded solutions represented with a different encoding technique (representation) (Aickelin and Bull, 2002; Lin *et al.*, 1994).

4. *Algorithm level*. This is the most general class. Each subpopulation can potentially run a somewhat different (evolutionary or even non-evolutionary) algorithm (Potts *et al.*, 1994; Sefrioui and Périaux, 2000; Tsutsui and Fujimoto, 1993).

Note that the algorithm-level heterogeneity contains all previous levels. For example, a dGA with different parameters in its subpopulations is also algorithm-level heterogeneous. We introduce it for hard-to-classify heterogeneous models. There also exist tools for the production of evolutionary algorithms not directly matching this classification, e.g., by allowing the automatic distribution of the computation (Arenas *et al*, 2002; Tierra, 2004), thus facilitating the creation of heterogeneous dGAs.

Another orthogonal level of heterogeneity can be defined with respect to the relationship maintained among the elementary algorithms in the dGA. Basically, if the amount of resources (individuals) of each subpopulation is not constant during the evolution, i.e., the size of a subpopulation depends on the previous success of its search strategy, then it can be said that subpopulations are competing. Otherwise, it seems that the subpopulations collaborate to find the optimum. Hence, we differentiate between competition-based heterogeneity (Hu and Goodman, 2002; Oh *et al.*, 2002; Schlierkamp-Voosen and Mühlenbein, 1996; Yi *et al.*, 2000) and collaboration-based heterogeneity (Herrera and Lozano, 2000; Venkateswaran *et al.*, 1996).

The models we deal with in this work (GD-RCGA, *Hy3*, and *Hy4*) exhibit different levels of heterogeneity. On the one hand, they are parameter-level heterogeneous, since the subpopulations use different values of selection pressure. But subpopulations also utilize different crossover operators, so they can also be considered as operator-level heterogeneous. On the other hand, the seminal GD-RCGA model shows collaboration-based heterogeneity, since its subpopulations cooperate, and do not compete, in order to perform the search.

3. *Hy4* Model

In this section, we describe the basic behavior of the GD-RCGA (Herrera and Lozano, 2000), and explain how it has been parallelized to yield the new *Hy3* algorithm (Alba *et al.*, 2004). Finally, the new *Hy4* algorithm is introduced and described.

3.1. GD-RCGA

The present availability of crossover operators for real-coded genetic algorithms (RCGAs) allows the possibility of using in the same algorithm different exploration or exploitation degrees, which leads to the design of heterogeneous distributed RCGAs based on this kind of operators (Herrera and Lozano, 1997). This issue is especially important for continuous optimization tasks. GD-RCGA is included into such a class of heterogeneous algorithms since it applies a different crossover operator in each of its component subpopulations. Figure 1 depicts a graphic outline of the algorithm.

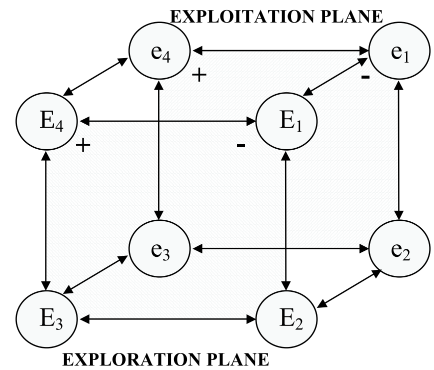


Fig. 1. Connection topology of a GD-RCGA.

The distribution scheme of GD-RCGA is based on a hypercube topology with three dimensions. There are two important faces in this hypercube that have to be considered:

- The *front side* is devoted to exploration. It is made up of four subpopulations $E_1, \ldots, E_4$, in which several exploratory crossovers are applied (see Table 2 in Section 3.1.3).

- The *rear side* promotes exploitation. It is composed of subpopulations $e_1, \ldots, e_4$, that apply exploitative crossover operators (see Table 2 in Section 3.1.3).

One salient feature of GD-RCGA is the use of an *elitist strategy* (de Jong, 1975) in the subpopulations, an important factor that may yield excessively rapid convergence. However, this is necessary in order to solve complex problems, because otherwise the best individual so far could disappear due to crossover or mutation.

The resulting algorithm is a parallel-suited multi-resolution method using several crossover operators which allow GD-RCGA to achieve simultaneously a diversified search (reliability), and an effective local tuning (accuracy). Furthermore, subpopulations are adequately connected for exploiting the multi-resolution in a *gradual* way, since the migrations between subpopulations belonging to different categories (front-rear migrations) may induce the refinement/expansion of the best emerging zones.

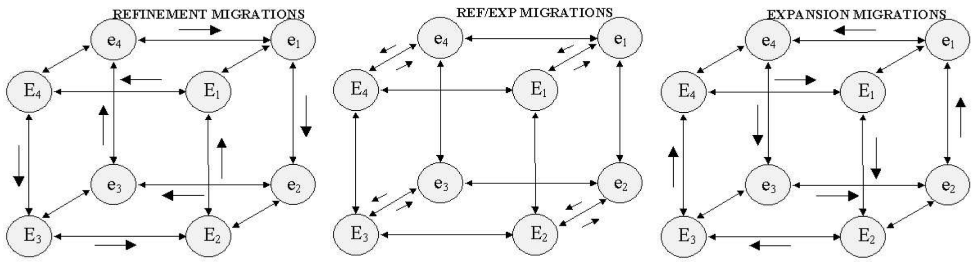


Fig. 2. Three types of migration in a GD-RCGA.

Let us explain the migration scheme and the selection mechanism used in GD-RCGA that help to establish a correct coordination between refinement and expansion.

3.1.1. Migration Scheme

Distributed GAs behavior is strongly determined by the migration mechanism (Alba and Troya, 2000; Cantú-Paz, 1995; Goldberg *et al.*, 1995). GD-RCGA uses a migration model where copies of migrants are sent only towards immediate neighbors along a dimension of the hypercube, and each subsequent migration takes place along a different dimension of the hypercube. Particularly, the best element of each subpopulation is sent towards the corresponding subpopulation periodically, as shown in Fig. 2. The sequence of applications is as follows: first, the refinement migrations; second, the refinement/expansion migrations; third, the expansion migrations; and finally, the sequence starts again. The place of an emigrant is taken by the incoming individual.

This migration scheme keeps a global elitist strategy, since the best element of all subpopulations is never lost, although it could be moved from one subpopulation to another.

3.1.2. Selection Mechanism

We use the same selection mechanism as in (Herrera and Lozano, 2000): *linear ranking selection* (Baker, 1985). It is used because the induced pressure can be easily adjusted. In the linear ranking selection, the individuals are sorted in order of raw fitness, and then the selection probability, p_s, of each individual I_i is computed according to its rank $\mathrm{rank}(I_i)$, with $\mathrm{rank}(I_{\mathrm{best}}) = 1$, by using the following non-increasing assignment function:

$$p_s(I_i) = \frac{1}{N}\left(\eta_{\max} - (\eta_{\max} - \eta_{\min})\frac{\mathrm{rank}(I_i) - 1}{N - 1}\right),$$

where N is the population size, and $\eta_{\min} \in [0, 1]$ specifies the expected number of copies for the worst individual (the best one has $\eta_{\max} = 2 - \eta_{\min}$ expected copies). The selection pressure of linear ranking is determined by $\eta_{\min}$. If $\eta_{\min}$ is low, high pressure is achieved, whereas if it is high, the pressure is low. Different selection pressure degrees were assigned to every subpopulation of GD-RCGAs, by selecting the $\eta_{\min}$ values as shown in Table 1.

Table 1. Crossover exploration/exploitation degrees and $\eta_{\min}$ values for each island.

	Exploitation				Exploration			
	e_4	e_3	e_2	e_1	E_1	E_2	E_3	E_4
Crossover	+		$\longleftarrow$	—	—		$\longrightarrow$	+
$\eta_{\min}$	0.8	0.7	0.6	0.5	0.3	0.2	0.1	0.0

Linear ranking is combined with *stochastic universal sampling* (Baker, 1987). This procedure guarantees that the number of copies of any individual is bounded by the floor and ceiling of its expected number of copies.

3.1.3. Fuzzy Connectives-Based Crossover Operators

The GD-RCGA was implemented endowed with a *fuzzy connective-based* crossover operator, called FCB-crossover, which is based on three functions, F, S, and M (Herrera *et al.*, 1995), for combining genes. Each function has different exploration/exploitation degree properties (Table 1) and can be calculated by four families of fuzzy connectives: *Logical*, *Hamacher*, *Algebraic*, and *Einstein*. Therefore, four families of FCB-crossover operators may be obtained by using these families of fuzzy connectives. As we stated before, these crossover operators have different properties: the F- and S-crossover operators inducing promoting exploration, and the M-crossover operators show exploitation. The exploration/exploitation degree of each crossover operator depends on the fuzzy connective on which it is based. In fact, the Einstein F- and S-crossovers show the maximum exploration, whereas the Logical ones present the minimum exploration. On the other hand, the Logical M-crossover shows the maximum level of exploitation. They were properly configured, as shown in Table 2.

Table 2. FCB-crossover configuration.

Rear Side	M-crossover	Front Side	F- and S-crossover
e_1	Hamacher	E_1	Logical
e_2	Algebraic	E_2	Hamacher
e_3	Einstein	E_3	Algebraic
e_4	Logical	E_4	Einstein

3.2. GD-RCGA Parallelization: the *Hy3* Model

Although GD-RCGA suggests a direct parallel implementation, its synchronous behavior has not been tested on parallel machines until (Alba *et al.*, 2003), where an asynchronous modification is also analyzed.

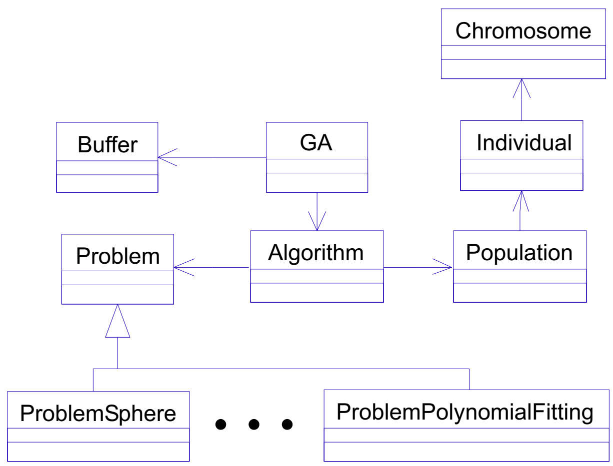


Fig. 3. UML design used to implement *Hy3*.

The *Hy3* model was implemented in Java, using sockets for inter-process communication. The UML class diagram in Fig. 3 depicts the design followed for its parallelization. The *GA* class represents the island behavior of the dGA. It manages the program execution by controlling the termination, the computation of new algorithm steps, and the migration scheme. The inclusion of the *Buffer* class was necessary to avoid deadlocks (they are uncoupling buffers). Each *GA* object has an associated *Buffer* element located in the same node, from which it takes migrated individuals. The rest of the classes are devoted to the computation of the problem, implementing selection, crossover, mutation, and evaluation of individuals.

Two versions of *Hy3* were implemented: *Synchronous Hy3* and *Asynchronous Hy3*. In the first one, in each migration phase, every subpopulation sends its best individual and then waits for another coming from the corresponding neighbor. In the async mode, this consideration is not taken into account; thus, any individual stored in its buffer by a precedent migration can be included in

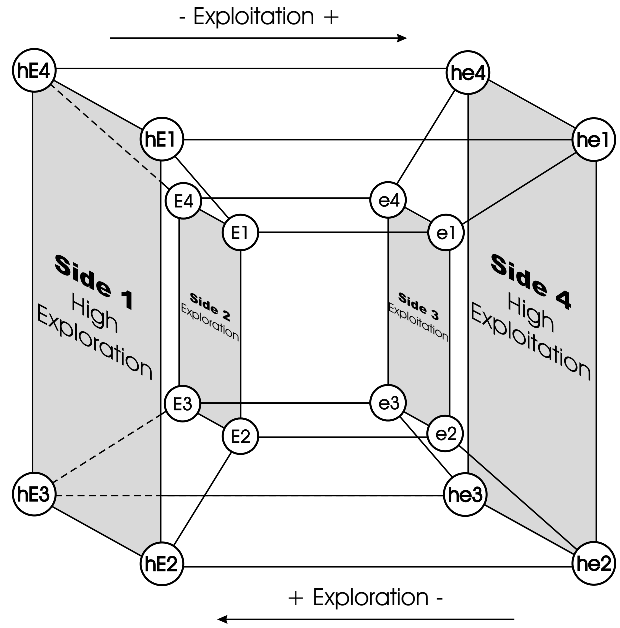


Fig. 4. *Hy4* topology.

the population at any time. Since the incoming individual always replaces the best one in the subpopulation in the two modes, the async version could lead to the loss of the global elitism due to its asynchronism (the best individual of a subpopulation could be replaced before it is migrated).

Hy3 was analyzed in (Alba *et al.*, 2003; 2004), and showed a high accuracy and parallel advantage, especially in asynchronous execution. However, we felt that we could go further with a new model in terms of either accuracy or efficiency. This leads us to propose the *Hy4* algorithm.

3.3. Extending *Hy3*: the *Hy4* Model

The *Hy4* model we are introducing here is aimed at reducing even more the frequency of convergence to local optima achieved by its two predecessors. The main goal of the method is to reach a new balance between exploration and exploitation that allows us to effectively tackle more complex problems. The *Hy4* algorithm is a heterogeneous dGA based on a hypercube topology with four dimensions (Fig. 4).

Four sides can be differentiated in the topology of the new model. *Side 2* and *Side 3* correspond to the *front side* and the *rear side* of *Hy3*. (Although they appear rotated with respect to Fig. 1.) Furthermore, these sides are configured as an *Hy3* algorithm (see Section 3.1): *Side 2* is devoted to exploration while *Side 3* focuses on exploitation. *Side 1* and *Side 4* are configured to stress the *Side 2* and *Side 3* search features, i.e., to achieve high exploitation and high exploration degrees, respectively.

If we consider the *Hy4* algorithm as two *Hy3* algorithms adequately connected, two cubes can be differentiated in Fig. 4: an inner one composed of *Side 2* and *Side 3*, and an outer one made of *Sides 1* and *Side 4*. These two cubes have the same selection mechanism, crossover operator configuration, and mutation as the *Hy3* model.

The different search features that we mentioned above are achieved by increasing the probabilities of the crossover (high exploration) and mutation (high exploitation) genetic operators in the respective subpopulations (see Section 5). Thus, the high exploration properties of *Side 1* are produced by higher crossover rates in subpopulations $hE1, \ldots, hE4$ whereas, by increasing the mutation rates in subpopulations $he1, \ldots, he4$, *Side 4* reaches the required high levels of exploitation. Additionally, these crossover and mutation rates are uniformly chosen so that the migrations between subpopulations provoke the refinement or expansion of the best zones.

The refinement is induced when migrations are produced:

- from an exploratory subpopulation toward an exploitative one, i.e., from hE_i to he_i,
- between two exploratory subpopulations from a higher degree to a lower one, i.e., from hE_{i+1} to hE_i, or
- between two exploitative subpopulations from a lower degree to a higher one, i.e., from he_i to he_{i+1}.

On the other side, the migrations in the opposite directions produce the expansion effect, since the individuals included may act as reference points for generating diversity in zones showing promising properties located in the exploitation planes.

Topology is an important factor in the performance of dGAs because it determines the speed at which a good solution spreads to other subpopulations. If the topology has a dense connectivity, a short diameter, or both, good solutions will spread quickly to all of the subpopulations (Cantú-Paz, 1995). Note that this kind of topology (along with frequent migrations) could lead dGAs to panmixia. In *Hy4*, this is avoided by performing migrations infrequently. Therefore, the short diameter of the *Hy4* model is suitable for favoring refinement and expansion, since genetic material will be quickly exchanged between subpopulations.

Figure 5 depicts the proposed migration model for the *Hy4* algorithm. Its goal is to maintain the gradual

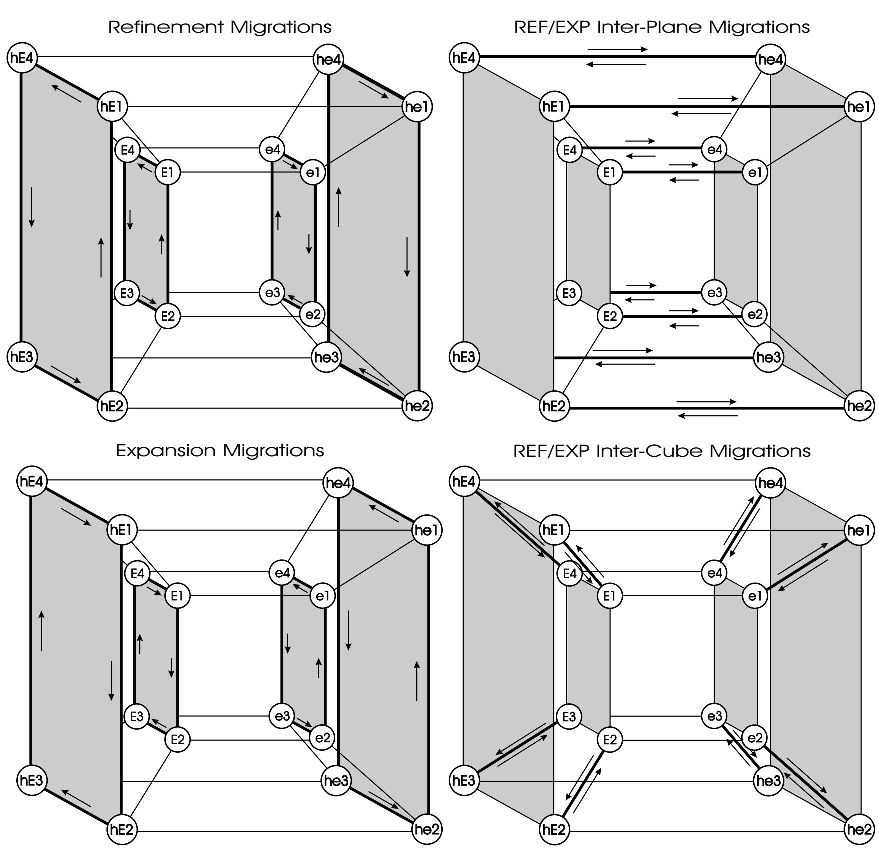


Fig. 5. Migration phases of *Hy4*.

effects of refinement and expansion of the *Hy3* migration presented in Section 3.1.1. Here, copies of the best individuals of each subpopulation are also sent towards the corresponding subpopulation, being their position filled by the incoming ones. Since we have an additional dimension with respect to the *Hy3* algorithm, four migration phases are needed. They are cyclicly applied in the following order: first, the refinement migrations; second, a first refinement/expansion phase (called *REF/EXP Inter-Plane Migration* in Fig. 5) takes place; third, the expansion migrations. (Note that these three phases correspond to the migration mechanism of the *Hy3* algorithm but they occur both in the inner and the outer cubes that form the *Hy4* model.) In the fourth place, the refinement/expansion *Inter-Cube* phase produces a genetic material exchange between subpopulations of the two cubes.

To finish the presentation of *Hy4*, we also want to point out that two versions of the model were implemented: a synchronous *Hy4* and an asynchronous *Hy4*. The basic aim is as for *Hy3* (cf. Section 3.2).

4. Problems

In this section, we present the benchmark used to test our algorithms. We have analyzed the results of minimization experiments on six test functions and three real-world problems in order to sustain our claims on a wide and diversified benchmark. It is usual to utilize these problems as a kind of standard benchmarking tasks for new algorithms, especially in continuous optimization. We have therefore selected the same benchmark as (Herrera and Lozano, 2000), because it is very complete, and also for comparison purposes. The problems are described in the next subsections.

4.1. Test Functions

We considered six classical and well-known test functions: *sphere* model (f_{Sph}) (de Jong, 1975; Schwefel, 1981), *generalized Rosenbrock's* function (f_{Ros}) (de Jong, 1975), *Schwefel's problem 1.2* (f_{Sch}) (Schwefel, 1981), *generalized Rastrigin's* function (f_{Ras}) (Bäck, 1992; Töorn and Antanas, 1989), *Griewangk's* function (f_{Gri}) (Griewangk, 1981), and *expansion of f10* (ef_{10}) (Whitley *et al.*, 1995). Figures 6 and 7 show their formulation and display their fitness landscape. The dimension of the search space is 10 for ef_{10} and 25 for the remaining test functions. Each has its particular features and difficulties:

- f_{Sph} is a continuous, strictly convex, and unimodal function.
- f_{Ros} is a continuous, nonseparable (nonlinear interactions among variables), and unimodal function, with the optimum located in a steep parabolic valley with a flat bottom (i.e., hard progress to the optimum).
- f_{Sch} is a continuous and unimodal function. Its difficulty concerns the fact that searching along the coordinate axes only gives a poor rate of convergence because the gradient of f_{Sch} is not oriented along the axes. It presents similar difficulties to f_{Ros}, but its valley is much narrower.
- f_{Ras} is a scalable, continuous, and multimodal function, which is made from f_{Sph} by modulating it with $a\cos(\omega x_i)$.
- f_{Gri} is a continuous and multimodal function. This function is difficult to optimize because it is nonseparable.
- f_{10} is a function that has nonlinear interactions between two variables. Its expanded version ef_{10} is built in such a way that it induces nonlinear interactions across multiple variables. It is nonseparable as well.

4.2. Real-World Problems

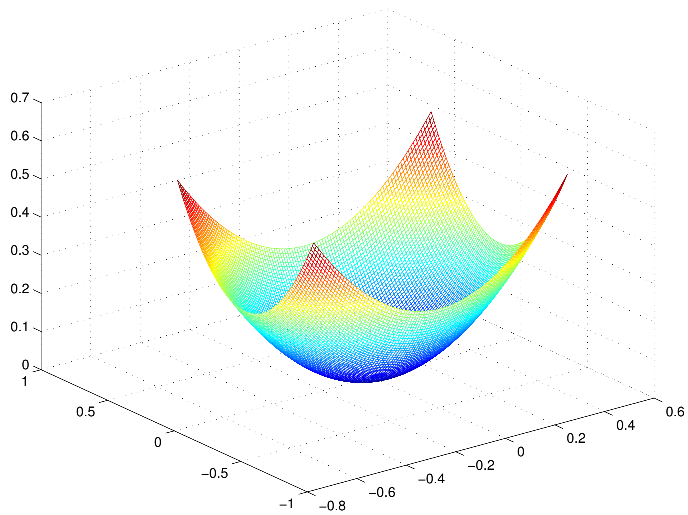
In order to better assess our conclusions, we chose three additional real-world problems: *systems of linear equations* (Eshelman *et al.*, 1997), *frequency modulation sound parameter identification problem* (Tsutsui and Fujimoto, 1993), and a *polynomial fitting problem* (Storn and Price, 1995). They all are described in the next subsections.

4.2.1. Systems of Linear Equations

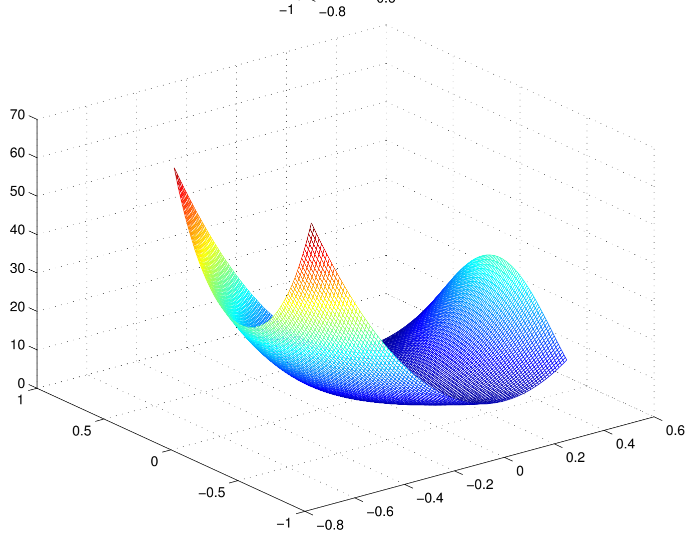
This problem may be stated as solving for the elements of a vector $\vec{x}$, given the matrix A and the vector $\vec{b}$ in the expression $A\vec{x} = \vec{b}$. The evaluation function used for these experiments is

$$f_{sle}(\vec{x}) = \Big|\sum_{i=1}^{n}\sum_{j=1}^{n}(a_{ij}x_j) - b_i\Big|.$$

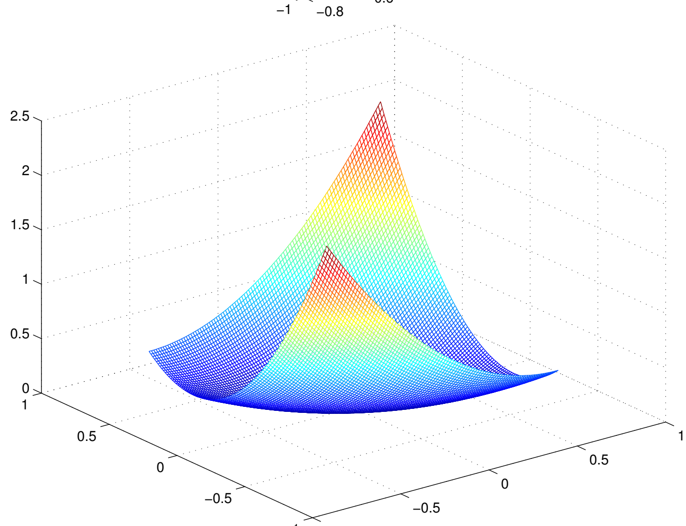
Clearly, if the system of equations is solvable, the best value for this objective function is $f_{sle}(\vec{x}^*) = 0$. Furthermore, the range of parameters is $[-9.0, +11.0]$. Interparameter linkage (i.e., nonlinearity) is easily controlled in systems of linear equations like this one, since their nonlinearity does not deteriorate as the number of parameters used is increased, and they proved to be quite difficult. We considered a ten-parameter problem instance. Its matrices are included in Fig. 8.



$$f_{Sph}(\vec{x}) = \sum_{i=1}^{n} x_i^2$$
$$-5.12 \leq x_i \leq +5.12$$
$$f_{Sph}(x^*) = 0$$

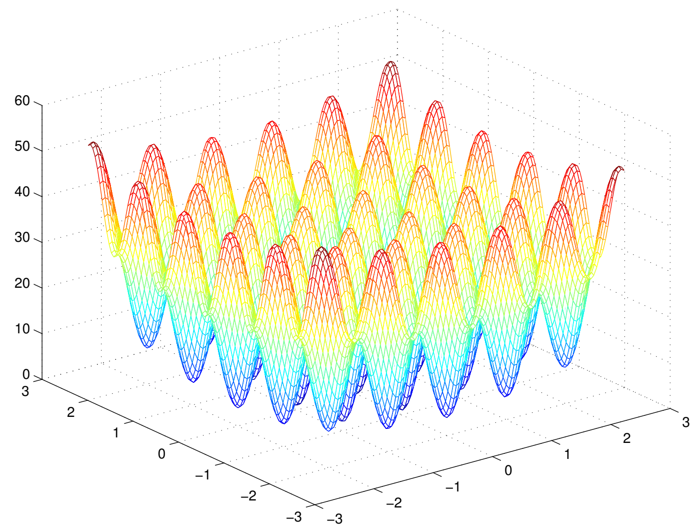


$$f_{Ros}(\vec{x}) = \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$$
$$-5.12 \leq x_i \leq +5.12$$
$$f_{Ros}(x^*) = 0$$

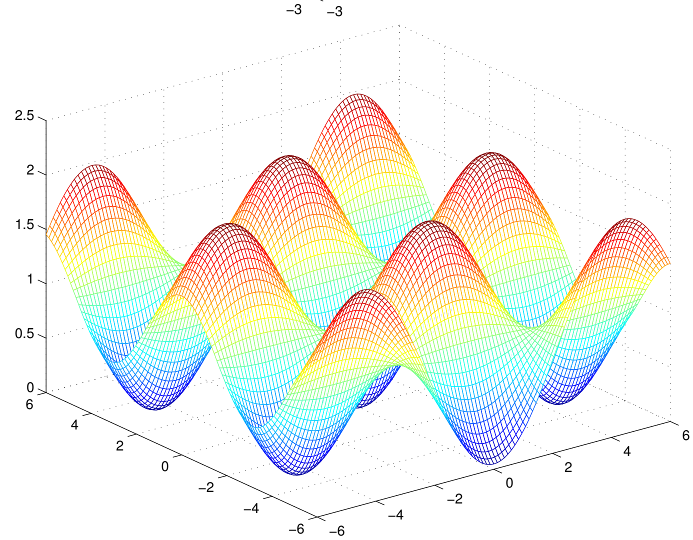


$$f_{Sch}(\vec{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$$
$$-65.536 \leq x_i \leq +65.536$$
$$f_{Sch}(x^*) = 0$$

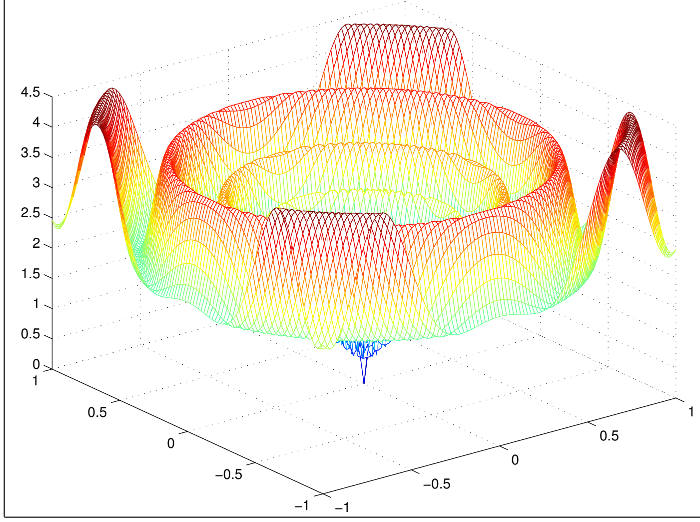
Fig. 6. Test functions (I).



$$f_{Ras}(\vec{x}) = an + \sum_{i=1}^{n}(x_i^2 - a\cos(\omega x_i))$$
$$a = 10, \omega = 2\pi, -5.12 \le x_i \le +5.12$$
$$f_{Ras}(x^*) = 0$$



$$f_{Gri}(\vec{x}) = 1 + \frac{1}{d}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)$$
$$d = 4000, -600.0 \le x_i \le +600.0$$
$$f_{Gri}(x^*) = 0$$



$$ef_{10}(\vec{x}) = f_{10}(x_n, x_1) + \sum_{i=1}^{n-1} f_{10}(x_i, x_{i+1})$$
$$f_{10}(x, y) = (x^2 + y^2)^{0.25} \times \left(\sin^2(50(x^2+y^2)^{0.1}) + 1\right)$$
$$x, y \in (-100, +100]$$
$$ef_{10}(x^*) = 0$$

Fig. 7. Test functions (II).

$$\begin{bmatrix} 5 & 4 & 5 & 2 & 9 & 5 & 4 & 2 & 3 & 1 \\ 9 & 7 & 1 & 1 & 7 & 2 & 2 & 6 & 6 & 9 \\ 3 & 1 & 8 & 6 & 9 & 7 & 4 & 2 & 1 & 6 \\ 8 & 3 & 7 & 3 & 7 & 5 & 3 & 9 & 9 & 5 \\ 9 & 5 & 1 & 6 & 3 & 4 & 2 & 3 & 3 & 9 \\ 1 & 2 & 3 & 1 & 7 & 6 & 6 & 3 & 3 & 3 \\ 1 & 5 & 7 & 8 & 1 & 4 & 7 & 8 & 4 & 8 \\ 9 & 3 & 8 & 6 & 3 & 4 & 7 & 1 & 8 & 1 \\ 8 & 2 & 8 & 5 & 3 & 8 & 7 & 2 & 7 & 5 \\ 2 & 1 & 2 & 2 & 9 & 8 & 7 & 4 & 4 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 40 \\ 50 \\ 47 \\ 59 \\ 45 \\ 35 \\ 53 \\ 50 \\ 55 \\ 40 \end{bmatrix}$$

Fig. 8. Matrices of the analysed system of linear equations.

4.2.2. Frequency Modulation Sound Parameter Identification Problem

The problem is to specify six parameters, a_1, ω_1, a_2, ω_2, a_3, ω_3, of the frequency modulation sound model represented by

$$y(t) = a_1 \sin\Big(\omega_1 t\theta + a_2 \sin\big(\omega_2 t\theta + a_3 \sin(\omega_3 t\theta)\big)\Big)$$

with $\theta = 2\pi/100$. If $\vec{x} = [a_1, \omega_1, a_2, \omega_2, a_3, \omega_3]$, the fitness function is defined as the sum of the squared errors between the evolved data and the model data as follows:

$$f_{fms}(\vec{x}) = \sum_{t=0}^{100} \big(y(t) - y_0(t)\big)^2,$$

where the model data are given by the following equation:

$$y_0(t) = 1.0 \sin\Big(5.0t\theta - 1.5 \sin\big(4.8t\theta + 2.0 \sin(4.9t\theta)\big)\Big).$$

Each parameter is in the range $[-6.4, +6.35]$. This is a highly complex multimodal problem having strong epistasis, with minimum value $f_{fms}(\vec{x}^*) = 0$.

4.2.3. Polynomial Fitting Problem

This problem lies in finding the coefficients of the following polynomial in z:

$$P(z) = \sum_{j=0}^{2k} c_j z^j, \quad k \in \mathbb{Z}^+$$

such that, $\forall z \in [-1, +1]$,

$$P(z) \in [-1, +1], \quad P(+1.2) \geq T_{2k}(+1.2),$$

$$P(-1.2) \geq T_{2k}(-1.2),$$

where $T_{2k}(z)$ is a Chebyshev polynomial of degree $2k$.

The solution to the polynomial fitting problem consists of the coefficients of $T_{2k}(z)$. This polynomial oscillates between -1 and $+1$ when its argument z is between -1 and $+1$. Outside this region, the polynomial rises steeply in the direction of high positive ordinate values. This problem has its roots in electronic filter design, and it challenges an optimization procedure by forcing it to find parameter values with grossly different magnitudes, something that is very common in industrial systems. The Chebyshev polynomial employed here is

$$T_8(z) = 1 - 32z^2 + 160z^4 - 256z^6 + 128z^8.$$

It is a nine-parameter problem ($\vec{x} = [x_1, \ldots, x_9]$). A small correction is needed in order to transform the constraints of this problem into an objective function to be minimized, called f_{Cheb} (see (Herrera and Lozano, 2000) for all the details). Each parameter (coefficient) is in the range $[-5.12, +5.12]$. The objective function value of the optimum is $f_{Cheb}(\vec{x}^*) = 0$.

5. Parameterization

In order to perform subsequent comparisons, the whole population of all the evaluated models is composed of 160 individuals. Thus, the *Hy3* variant uses 20 individuals per subpopulation, whereas the islands of the *Hy4* algorithms contain 10 individuals. All the models also perform a migration every 160 generations. The mutation operator applied is *nonuniform* mutation (Michalewicz, 1992). This operator needs two parameters: b (set to value 5), which determines the degree of dependency on the number of iterations, and MG, which is the maximum number of generations (see its different values in Table 4).

Table 3. Selective pressure ($\eta_{\min}$), crossover rate, and mutation rate for *Hy4* islands.

	Exploitative side				Explorative side			
	he_4	he_3	he_2	he_1	hE_1	hE_2	hE_3	hE_4
$\eta_{\min}$	0.8	0.7	0.6	0.5	0.3	0.2	0.1	0.0
Crossover	0.6	0.6	0.6	0.6	0.7	0.8	0.9	1.0
Mutation	0.5	0.4	0.3	0.2	0.125	0.125	0.125	0.125

The *Hy3* model and the subpopulations $e_1, \ldots, e_4$ and $E_1, \ldots, E_4$ of the *Hy4* algorithms present the same selective pressure, crossover, and mutation configuration. All of them use a crossover probability of 0.6 and a mutation probability of 0.125. Concerning the selective pressure of linear ranking selection, they use the $\eta_{\min}$ values shown in Table 1. The rest of the *Hy4* islands, i.e., $he_1, \ldots, he_4$ and $hE_1, \ldots, hE_4$ follows the specific configuration shown in Table 3. This configuration allows

Table 4. Maximum number of generations (MG) and target fitness (TF).

Problem	MG	TF	Problem	MG	TF
f_{Sph}	15000	2e-13	f_{Ras}	30000	4e-11
f_{Ros}	60000	9e0	f_{Gri}	5000	2e-2
f_{Sch}	5000	4e0	ef_{10}	15000	2e-3

Problem	MG	TF
f_{sle}	5000	4e1
f_{fms}	5000	1e1
f_{Cheb}	100000	2e2

us to emphasize the explorative and exploitative features of the new faces introduced by the *Hy4* model. In the appendix, we give a detailed description of the *Hy4* configuration.

The original GD-RCGA work imposed a predefined number of iterations (5000), but we cannot do the same because we want to measure the time to find equivalent solutions with *Hy3*/*Hy4* models (sync/async versions) and also to compute with respect to the original work. Thus, we calculated a maximum number of iterations for every problem (see Table 4), and we defined our goal as reaching the fitness values appearing in this table (that correspond to the average of the best fitness function found in the basic reference work (Herrera and Lozano, 2000)). The presented results are the averages over 30 independent runs, all of them reaching the target fitness in Table 4.

Our computing system is a cluster of Sun Ultra 1 workstations running Solaris 2.8. Each of them has a 400 MHz Ultra-SPARC II processor with 256 MB of memory. The machines are interconnected by a Fast-Ethernet network at 100 Mbps. We used JDK 1.4.0-b92 and compiled the programs with the `-O` optimization flag.

6. Results

Let us now proceed with the presentation of the results. We first analyze the sync/async behavior with respect to the execution time and numerical effort of the models. In a later subsection, a comparison between *Hy3* and *Hy4* is performed.

6.1. Run Time Results

In Table 5 we show the execution time of synchronous and asynchronous versions of the *Hy3* model. We consider first the monoprocessor case. It can be observed that the synchronous algorithms produced a faster execution than the asynchronous ones. In fact, we can notice that there exists statistical confidence for this claim (see the

Table 5. Execution times (in ms) of the synchronous and asynchronous *Hy3*.

Time (ms)	1 CPU			8 CPUs		
	Sync	Async	t-test	Sync	Async	t-test
f_{Sph}	51826	60221	+	9115	7890	+
f_{Ros}	28173	111638	+	4797	8150	–
f_{Sch}	9670	12952	+	1729	1956	–
f_{Ras}	111367	121567	+	16867	16073	–
f_{Gri}	10344	17533	+	1879	2339	+
ef_{10}	54215	62840	+	8982	8710	+
f_{sle}	1123	1104	–	563	566	–
f_{fms}	8894	10353	–	1714	1612	–
f_{Cheb}	8863	8935	–	1430	11436	–

"+" symbols meaning the significance of the t-test at the 95% level) for the six test functions. When using a single processor, the original idea of synchronous execution of the underlying model seems to perform well. Similar times were reported for the three complex instances for the two versions of *Hy3*. Although this is somewhat surprising, we can check in the right part of Table 5 that the sync and async differences vanish when running the eight subalgorithms in eight CPUs. These results can be explained because of the very fast optimization achieved for the test functions, in which fitness evaluation is extremely fast and therefore residual times (setting up processes, delays in communications, etc.) dominate the whole execution time.

Therefore, we conclude that the run times provided by the *Hy3* model are relatively independent of the synchronization mechanism, because of its multi-migration scheme. However, we do report a great improvement in the parallel efficiency (η) of the asynchronous models with respect to the synchronous ones if N is the number of processors and s_N is the speedup ($s_N = \bar{t}_1/\bar{t}_N$), with $\eta = s_N/N$ being the efficiency. In Fig. 9 we include the Δ_η value as the difference between the parallel efficiency of async and sync executions, i.e., $\Delta_\eta = \eta_{\text{async}} - \eta_{\text{sync}}$. A positive value of Δ_η means an improved result of async *Hy3* versus the sync one, while a negative value of this measure points out a higher efficiency of the sync version. One can notice that all but one value are positive and of a large magnitude, meaning that the efficiency (and thus scalability and quality of the parallelization) is really higher in the asynchronous case. For f_{sle} the efficiency remains almost the same, and f_{Cheb} is an exception, since there exists a huge variance in the time and evaluations to find a solution for this problem.

Table 6 presents the execution times of the *Hy4* model. In the monoprocessor case, the behavior of this

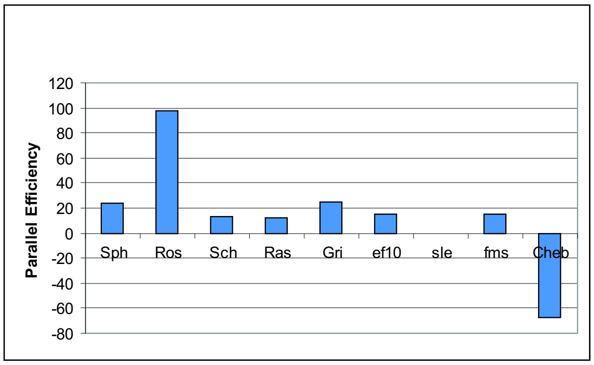


Fig. 9. Parallel efficiency of the *Hy3* model.

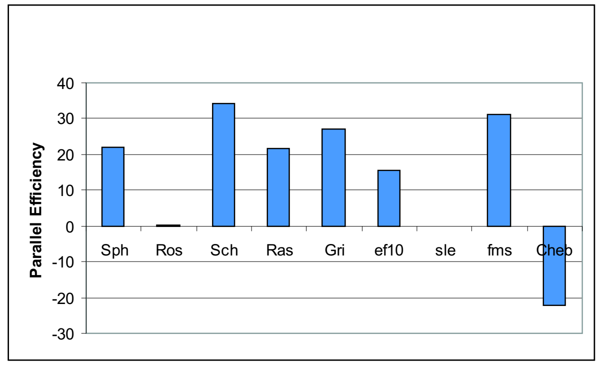


Fig. 10. Parallel efficiency of the *Hy4* model.

Table 6. Execution times (in ms) of the synchronous and asynchronous *Hy4* models.

Time (ms)	1 CPU			16 CPUs		
	Sync	Async	t-test	Sync	Async	t-test
f_{Sph}	91223	82433	+	10995	6991	+
f_{Ros}	52975	87766	–	2832	4686	–
f_{Sch}	11632	18060	+	1956	1583	–
f_{Ras}	183071	172358	–	21032	14188	+
f_{Gri}	16390	20275	+	2507	1864	+
ef_{10}	88054	94188	–	8701	7466	+
f_{sle}	2364	2465	–	1195	1195	–
f_{fms}	16189	20929	+	1984	1592	+
f_{Cheb}	19630	11411	–	1709	1434	–

algorithm is similar to the *Hy3* one, i.e., sync versions are faster than async ones. Nevertheless, there is an exception for f_{Sph} (with statistical confidence) in which the asynchronous *Hy4* is faster than the synchronous one. The results reported for the parallel execution (right half of Table 6) over 16 CPUs exhibit the same behavior as the commented exception. The reason is that the synchronization constraints among 16 islands imposed by the sync *Hy4* penalize the execution time. It is important to note that the natural execution mode of *Hy3* and *Hy4* uses 8 and 16 CPUs, respectively. The Δ_η value for the parallel executions is also displayed in Fig. 10. It shows, as it happened with the *Hy3* model, that the asynchronous *Hy4* parallelization scales better than the sync one, too.

6.2. Numerical Effort

Now, we turn to the analysis of the number of evaluations, i.e., the numerical effort to solve the problems. The results of the *Hy3* and *Hy4* models are shown in Tables 7 and 8, respectively. Overall, it seems that the two versions of the two algorithms need a similar effort to solve all the optimization tasks, which is an expected result since all the machines have a similar computational power.

The numerical behavior is the same in 8 out of 9 problems, where *Hy3* and *Hy4* are similar (the "–" symbol), or we find one of them more efficient than the other depending on the number of CPUs. Hence, we can conclude nothing about the superiority of any version of the two models: they seem equally well suited and efficient for the problems considered. There is one exception in each model. In the *Hy3* algorithm, the sync version is always more efficient numerically for the function f_{Gri}. On the other hand, in the *Hy4* model, the async version needs a lower number of evaluations to solve the f_{Sph} function.

6.3. Comparison between the *Hy3* and *Hy4* Models

Let us now address an explicit comparison between the *Hy3* and *Hy4* models. Tables 9 and 10 summarize the execution times of the models when running over 1 CPU and 8/16 CPUs, respectively. The two tables include an additional new column, called *ratio*, containing the relation-

Table 7. Number of evaluations of synchronous and asynchronous *Hy3*.

Evals	1 CPU			8 CPUs		
	Sync	Async	t-test	Sync	Async	t-test
f_{Sph}	215505.1	219571.5	–	233929.1	212543.7	+
f_{Ros}	110389.5	389569.4	+	114053.1	211710.7	–
f_{Sch}	33933.5	41857.7	+	33965.8	41046.1	–
f_{Ras}	444465.8	423820.1	+	432104.0	429567.4	–
f_{Gri}	36526.1	55806.3	+	38614.1	53480.6	+
ef_{10}	226262.1	229478.1	–	238077.1	233348.5	+
f_{sle}	176.2	176.3	–	176.0	176.9	–
f_{fms}	12759.5	14391.7	–	15728.9	15444.2	–
f_{Cheb}	6227.8	6059.3	–	5551.1	65007.7	–

Table 8. Number of evaluations of synchronous and asynchronous *Hy4*.

Evals	1 CPU			16 CPUs		
	Sync	Async	t-test	Sync	Async	t-test
f_{Sph}	437272.6	385450.7	+	443785.7	392144.1	+
f_{Ros}	235759.1	384701.5	–	106783.8	240906.5	+
f_{Sch}	42180.7	69277.6	+	49662.9	47212.4	–
f_{Ras}	828287.0	764227.0	+	835929.7	798764.4	–
f_{Gri}	61970.1	79646.8	+	86087.4	69430.7	+
ef_{10}	424349.1	428604.6	–	460939.9	413389.7	–
f_{sle}	198.6	196.7	–	198.1	198.4	–
f_{fms}	24153.3	30142.9	+	29322.5	24392.7	–
f_{Cheb}	13906.2	6826.6	–	12450.1	9763.3	–

Table 9. Time (in ms) of the *Hy3* and *Hy4* models running over 1 CPU.

Time (ms)	Sync				Async			
	Hy3	*Hy4*	Ratio	t-test	*Hy3*	*Hy4*	Ratio	t-test
f_{Sph}	51826	91223	1.76	+	60221	82433	1.36	+
f_{Ros}	28173	52975	1.88	+	111638	87766	0.79	–
f_{Sch}	9670	11632	1.20	–	12952	18060	1.39	–
f_{Ras}	111367	183071	1.64	–	121567	172358	1.42	–
f_{Gri}	10344	16390	1.58	–	17533	20275	1.16	+
ef_{10}	54215	88054	1.62	+	62840	94188	1.50	+
f_{sle}	1123	2364	2.11	–	1104	2465	2.23	–
f_{fms}	8894	16189	1.82	–	10353	20929	2.02	–
f_{Cheb}	8863	19630	2.21	–	8935	11411	1.28	+

ship between the *Hy4* and *Hy3* execution times. A ratio larger than 1 means that the *Hy4* model is slower than the *Hy3* one. Analogously, if this ratio is smaller than 1, then the *Hy4* model is faster than the *Hy3* one.

The results presented in Table 9 show that, for almost all the problems, *Hy3* is faster than *Hy4* when they are executed over 1 CPU (i.e., concurrently). This can be explained by the larger number of subpopulations of the *Hy4* algorithm: this produces a higher overload (e.g., context switches), even though the whole population size of the two models is the same. The asynchronous execution of the f_{Ros} function constitutes the unique exception. However, an additional significance test reveals that the times are statistically similar.

If we compare the synchronous parallel executions of the two models (the left part of Table 10), we can notice that the *Hy3* algorithm is faster than the *Hy4* one, which is somewhat surprising since the latter runs over twice the number of processors. This can be justified by two facts.

Table 10. Time (in ms) of the *Hy3* and *Hy4* models running over 8 and 16 CPU, respectively.

Time (ms)	Sync				Async			
	Hy3	*Hy4*	Ratio	t-test	*Hy3*	*Hy4*	Ratio	t-test
f_{Sph}	9115	10995	1.21	+	7890	6991	0.89	+
f_{Ros}	4797	2832	0.59	+	8150	4686	0.57	+
f_{Sch}	1729	1956	1.13	–	1956	1583	0.81	–
f_{Ras}	16867	21032	1.25	+	16073	14188	0.88	–
f_{Gri}	1879	2507	1.33	+	2339	1864	0.80	+
ef_{10}	8982	8701	0.97	+	8710	7466	0.86	+
f_{sle}	563	1195	2.12	–	566	1195	2.11	–
f_{fms}	1714	1984	1.16	–	1612	1592	0.99	–
f_{Cheb}	1430	1709	1.19	–	11436	1434	0.13	+

First, the number of individuals in 16 islands of *Hy4* is 10, in order to maintain a total population size of 160 individuals. Second, each subpopulation in *Hy4* has to synchronize with four other ones (its four neighbors in the four-dimensional hypercube) while in *Hy3* each population has only three neighbors. These two facts make the ratio “computation/communication” decrease and therefore the synchronization constraints induce larger execution times. Only the sync execution of the f_{Ros} and ef_{10} functions takes advantage of the higher computational power used by the *Hy4* algorithm.

In the right half of Table 10 we include a comparison of the asynchronous executions of the *Hy3* and *Hy4* models. This time, *Hy4* performs as expected, since it is faster than *Hy3* (see the ratio values lower than 1 in the table). This indicates that the asynchronous *Hy4* model enhances the scalability and the quality of the parallelization by avoiding the synchronization requirements imposed by the sync model. Special attention must be paid to the f_{Cheb} function, where the ratio is 0.13. This shows the capability of *Hy4* to improve the search process over *Hy3* with complex instances. For the rest of the test problems, the ideal ratio of $0.5 = 8/16$ (8 parallel subpopulations against 16 ones) is only approximated by the computation of the f_{Ros} function. Twice the number of islands plus half a population size of the *Hy4* model can explain this result (i.e., residual times control the total execution time).

To complete the section, we must discuss the actual benefits of the *Hy4* model of search: its ability to manage premature convergence due to an enhanced model of search based on a new balance between exploration and exploitation. Up to now, we have presented the average results over 30 independent runs. This time, we show the hit rate of the *Hy3* and *Hy4*. The hit rate is the percentage of executions that reached the target fitness. If we focus on the numerical aspect (i.e., the effort and hit

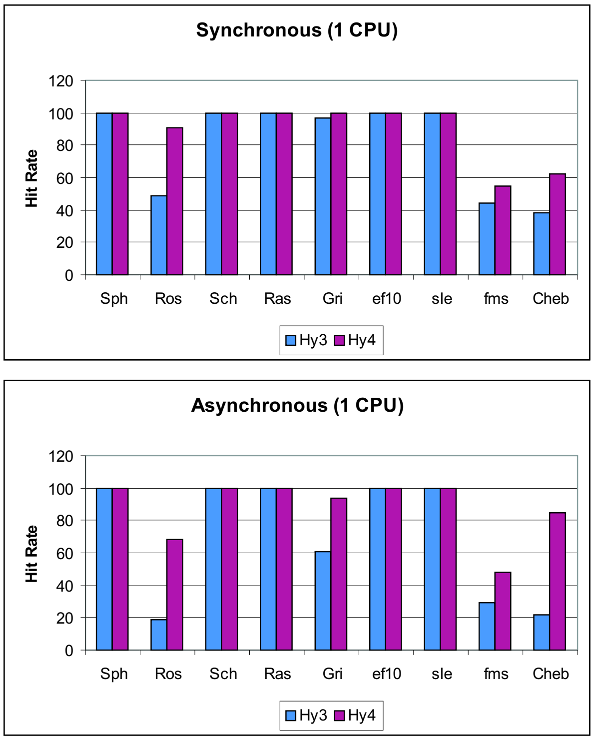


Fig. 11. Hit rate of the *Hy3* and *Hy4* models (1 CPU).

rate), we can analyze the results when the two models run over 1 CPU, which is presented in Fig. 11. Several important facts can be remarked from this figure. First, for all the solved instances, any *Hy4* version (sync/async) of the algorithm always obtains a higher hit rate than the *Hy3* one. This can be explained by the *Hy4* enhanced search model. Second, the synchronous *Hy4*, as a direct extension of the GD-RCGA model, shows the highest hit rates, justifying the contribution of this work. Third, a major improvement of the hit rate is emphasized if we consider the asynchronous versions of the two models (the bottom part of Fig. 11). As an example, we can point out the f_{Cheb} function, where the hit rate of async *Hy3* is around 20%, while the async *Hy4* one is over 80%. A subtle reason that explains this result is the implementation of the asynchronism. Subpopulations in the asynchronous versions of the models have to perform a polling operation to check its buffer for incoming individuals. This polling operation is accomplished every 30 iterations, so if it finds an empty buffer due to the asynchronism of the algorithm, the subpopulation does not incorporate any individual at least in 60 iterations, and therefore the new individual may be grossly incompatible with the target subpopulation (the "mule" effect) (Herrera and Lozano, 2000; Lin *et al.*, 1994). This leads the async models to evolve subpopulations during a larger number of isolated iterations (more exploitation) but, since *Hy4* has twice the number of islands of *Hy3*, this exploitation is carried out over more separated regions of the search space (i.e., more exploration). Thus, the new balance exploration/exploitation achieved increases the probability of finding a solution. Finally, we want to notice the suitability of the *Hy4* algorithm to the complex problem instances f_{Ros}, f_{fms}, and f_{Cheb}, for which this new model of search fairly improves the hit rate of the *Hy3* one.

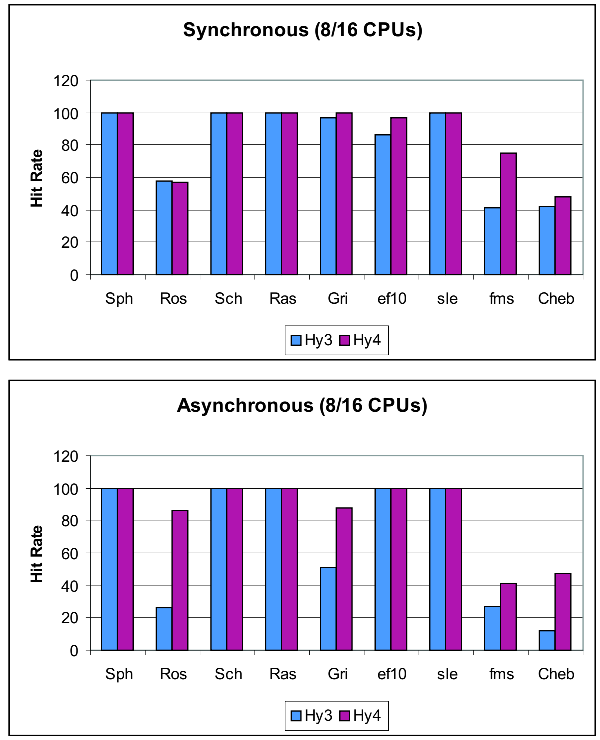


Fig. 12. Hit rate of *Hy3* (8 CPUs) and *Hy4* (16 CPUs).

If we analyze the hit rate when the two models run over 8/16 CPUs (Fig. 12), we can observe that the parallel *Hy4* execution also outperforms the *Hy3* one, which is an expected result since we have a homogeneous computing platform. Several conclusions may be drawn from this discussion about the hit rate of the algorithms:

- For our benchmark, the new *Hy4* model proposed outperforms the *Hy3* one.
- The improvement is more significant when comparing the asynchronous versions of the two models.
- The *Hy4* algorithm achieves higher hit rates for more complex functions.

7. Conclusions and Future Work

This paper presents the *Hy4* model of search, being an extension of the *Hy3* algorithm, which is in turn a physical parallelization of the gradual distributed real-coded GA. The basic underlying search model naturally provides a set of eight subpopulations residing in a cube topology having two faces for promoting exploration and exploitation. Furthermore, *Hy4* uses instead 16 islands arranged in a hypercube of four dimensions, where two new faces with different exploration/exploitation search capabilities are added to the *Hy3* algorithm. The exploration or exploitation degrees of the subpopulations belonging to the same side are also gradually configured in *Hy4*, thus obtaining a general and powerful parallel multi-resolution method for continuous optimization. With this model we also investigate the advantages provided by an asynchronous design versus a synchronous one. The motivation for this extended model is the larger accuracy of the original works for optimization problems coming from continuous domains in optimization.

We performed our analysis under the assumption that our algorithms must reach the same average solution as the one reported by the basic reference work. With this goal in mind, we solved nine problems. The results show that the asynchronous parallelization (*Hy3* and *Hy4*) can provide a larger efficiency for all the problems, which confirms other existing results like (Alba and Troya, 2001). This is justified because the synchronization constraints among a larger number of islands imposed by the sync version of the model penalize the execution time. However, this is in contrast with what generally happens in specialized distributed GAs, in which async versions are usually much faster that sync ones (Alba and Troya, 2001).

In order to show the larger accuracy of the *Hy4* model, we compared it with the *Hy3* one. This comparison was carried out with respect to the hit rate (the relation between the number of executions reaching the target fitness and the total number of performed tests). From this point of view, we can conclude that, for our benchmark, the new model clearly outperforms the *Hy3* algorithm. Furthermore, the improvement of *Hy4* is quite evident when comparing the asynchronous versions of the two models, and when they have to deal with complex functions. Thus, the goal of obtaining a new method that avoids the premature convergence problem was achieved by designing *Hy4*.

As a future work, we will apply the model to combinatorial optimization, i.e., to the discrete domain of optimization. Also, we plan to introduce a restart technique and new extended models based on the hypercubic topology of search to improve the results on even more complex problems.

Acknowledgments

This work was partly funded by the Ministry of Science and Technology and FEDER under the contracts TIC2002-04498-C05-02 (the TRACER project) and TIC2002-04309-C02-02.

References

Adamidis P. and Petridis V. (1996): *Co-operating populations with different evolution behaviors*. — Proc. 3rd IEEE Conf. *Evolutionary Computation*, New York, pp. 188–191.

Adamidis P. and Petridis V. (2002): *On modelling evolutionary algorithm implementations through co-operating populations*, In: Parallel Problem Solving from Nature (PPSN VII), (J.J. Merelo Guervós, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villacañas and H.-P. Schwefel, Eds.). — Granada, Spain: Springer, pp. 321–330.

Aickelin U. and Bull L. (2002): *Partnering strategies for fitness evaluation in a pyramidal evolutionary algorithm*. — Proc. Genetic and Evolutionary Comput. Conf. *GECCO'02*, New York, pp. 263–270.

Alba E., Luna E. and Nebro A.J. (2003): *Parallel heterogeneous genetic algorithms for continuous optimization*. — Int. Parallel and Distributed Proces. Symp. (IPDPS-NIDISC'03), Nice, France, p. 147.

Alba E., Luna F., Nebro A.J. and Troya J.M. (2004): *Parallel heterogeneous genetic algorithms for continuous optimization*. — Parallel Comput., (to appear).

Alba E., Nebro A.J. and Troya J.M. (2002): *Heterogeneous computing and parallel genetic algorithms*. — J. Parall, Distrib. Comput., Vol. 62, pp. 1362–1385.

Alba E. and Tomassini M. 2002): *Parallelism and evolutionary algorithms*. — IEEE Trans. Evolut. Comput., Vol. 6, No. 5, pp. 443–462.

Alba E. and Troya J.M. (1999): *A survey of parallel distributed genetic algorithms*. — Complexity, Vol. 4, No. 4, pp. 31–52.

Alba E. and Troya J.M. (2000): *Influence of the migration policy in parallel distributed GAs with structured and panmictic populations*. — Appl. Intell., Vol. 12, No. 3, pp. 163–181.

Alba E. and Troya J.M. (2001): *Analyzing synchronous and asynchronous parallel distributed genetic algorithms*. — Future Generat. Comput. Syst., Vol. 17, No. 4, pp. 451–465.

Arenas M.G., Collet P., Eiben A.E., Jelasity M., Merelo J.J., Paechter B., Preub M. and Schoenauer M. (2002): *A framework for distributed evolutionary algorithms*, In: Parallel Problem Solving from Nature (PPSN VII) (J.J. Merelo Guervós, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villacañas and H.-P. Schwefel, Eds.). — Granada, Spain: Springer, pp. 665–675.

Bäck T. (1992): *Self-Adaptation in Genetic Algorithms*. — Proc. 1st Europ. Conf. *Artificial Life*, Cambridge, MA, pp. 263–271.

Bäck T. (1996): *Evolutionary Algorithms in Theory and Practice*. — Oxford: Oxford University Press.

Bäck T., Fogel D.B. and Michalewicz Z. (1997): *Handbook of Evolutionary Computation*. — Oxford: Oxford University Press.

Baker J.E. (1985): *Adaptive selection methods for genetic algorithms*. — Proc. 1st Int. Conf. *Genetic Algorithms Appl.*, Hillsdale, NJ, pp. 101–111.

Baker J.E. (1987): *Reducing bias and inefficiency in the selection algorithm*. — Proc. 2nd Int. Conf. *Genetic Algorithms Appl.*, Hillsdale, NJ, pp. 14–21.

Cantú-Paz E. (1995): *A summary of research on parallel genetic algorithms*. — Tech. Rep. No. 95007, Univ. Illinois, Urbana-Champaign, Illinois GA Laboratory.

de Jong K.A. (1975): *An analysis of the behavior of a class of genetic adaptive Systems*. — Ph.D. thesis, Univ. Michigan, Ann Arbor.

Eshelman L.J., Mathias K.E. and Schaffer J.D. (1997): *Convergence controlled variation*, In: Foundations of Genetic Algorithms 4 (R. Belew and M. Vose, Eds.). — San Mateo, CA: Morgan Kaufmann, pp. 203–224.

Goldberg D.E. (1989): *Genetic Algorithms in Search, Optimization and Machine Learning*. — Boston: Addison-Wesley.

Goldberg D.E., Kargupta H., Horn J. and Cantú-Paz E. (1995): *Critical deme size for serial and parallel genetic algorithms*. — Tech. Rep. No. 95002, Univ. Illinois, Urbana-Campaign, Illinois Genetic Algorithms Laboratory.

Griewangk A.O. (1981): *Generalized descent of global optimization*. — J. Optim. Theory Appl., Vol. 34, No. 3, pp. 11–39.

Herrera F. and Lozano M. (1997): *Heterogeneous distributed genetic algorithms based on the crossover operator*. — Proc. 2nd IEE/IEEE Int. Conf. *Genetic Algorithms Eng. Syst.: Innovations Appl.*, Glasgow, UK, pp. 203–208.

Herrera F. and Lozano M. (2000): *Gradual distributed real-coded genetic algorithms*. — IEEE Trans. Evolut. Comput., Vol. 4, No. 1, pp. 43–63.

Herrera F., Lozano M. and Moraga C. (1998): *Hybrid distributed real-coded genetic algorithms*, In: Parallel Problem Solving from Nature (PPSN V) (A.E. Eiben, T. Back, M. Schoenauer and H-P. Schwefel, Eds.). — Amsterdam: Springer, pp. 879–888.

Herrera F., Lozano M. and Verdegay J.L. (1995): *Fuzzy connective based crossover operators to model genetic algorithms population diversity*. — Tech. Rep. No. DECSAI-95110, University of Granada, 18071 Granada, Spain.

Hinterding R., Michalewicz Z. and Peachey T.C. (1996): *Self-adaptive genetic algorithm for numeric functions*, In: Parallel Problem Solving from Nature (PPSN IV) (H.-M. Voigt, W. Ebeling, I. Rechenberger and H.-P. Schwefel, Eds.). — Berlin: Springer, pp. 420–429.

Hiroyasu T., Miki M. and Negami M. (1999): *Distributed genetic algorithms with randomized migration rate*. — Proc. IEEE Conf. *Systems, Man and Cybernetics*, Tokyo, Japan, Vol. 1, pp. 689–694.

Holland J.H. (1997): *Adaptation in natural and artificial systems*. — Ph.D. thesis, Univ. Michigan, Ann Arbor.

Hu J.J. and Goodman E.D. (2002): *The hierarchical fair competition (HFC) model for parallel evolutionary algorithms*. — Proc. Congress *Evolutionary Computation, CEC2002*, Honolulu, USA, pp. 49–54.

Lin S.-L., Punch III W.F. and Goodman E.D. (1994): *Coarse-grain parallel genetic algorithms: Categorization and new approach*. — Proc. 6th IEEE Symp. *Parallel and Distributed Processing*, Dallas, USA, , pp. 28–37.

Manderick B. and Spiessens P. (1989): *Fine-grained parallel genetic algorithms*. — Proc. 3rd Int. Conf. *Genetic Algorithms*, Virginia, USA, pp. 428–433.

Michalewicz Z. (1992): *Genetic Algorithms + Data Structures = Evolution Programs*. — Berlin: Springer.

Miki M., Hiroyasu T., Kaneko M. and Hatanaka K. (1999): *A parallel genetic algorithm with distributed environment scheme*. — Proc. IEEE Conf. *Systems, Man and Cybernetics*, Tokyo, Japan, pp. 695–700.

Mühlenbein H., Schomisch M. and Born J. (1991): *The parallel genetic algorithm as function optimizer*. — Proc. 4th Int. Conf. *Genetic Algorihtms*, San Mateo, CA, pp. 271–278.

Oh S.-K, Lee C.-Y. and Lee J.-J. *A new distributed evolutionary algorithm for optimization in nonstationary environments*. — Proc. Congr. *Evolutionary Computation*, Honolulu, USA, pp. 378–383.

Potts J.C., Giddens T.D. and Yadav S.B. (1994): *The development and evaluation of an improved genetic algorithm based on migration and artificial selection*. — IEEE Trans. Syst. Man Cybern., Vol. 24, No. 1, pp. 73–86.

Schlierkamp-Voosen D. and Mühlenbein H. (1994): *Strategy adaptation by competing subpopulations*, In: Parallel Problem Solving from Nature (PPSN III) (Y. Davidor, H.-P. Schwefel and R. Männer, Eds.). — Berlin: Springer, pp. 199–208.

Schlierkamp-Voosen D. and Mühlenbein H. (1996): *Adaptation of population sizes by competing subpopulations*. — Proc. Int. Conf. *Evolutionary Computation*, Nagoya, Japan, pp. 199–208.

Schnecke V. and Vornberger O. (1996): *An adaptive parallel algorithm for VLSI-layout optimization*, In: Parallel Problem Solving from Nature (PPSN IV) (H.-M. Voigt, W. Ebeling, I. Rechenberger and H.-P. Schwefel, Eds.). — Berlin: Springer, pp. 22–27.

Schwefel H.-P. (1981): *Numerical Optimization of Computer Models*. — Chichester: Wiley.

Sefrioui M. and Périaoux J. (2000): *A hierarchical genetic algorithm using multiple models for optimization*, In: Parallel Problem Solving from Nature (PPSN VI) (M. Schoenauer, Kalyanmoy Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo and H.-P. Schwefel, Eds.). — Paris: Springer, pp. 879–888.

Storn R. and Price K. (1995): *Differential evolution – A simple efficient adaptive scheme for global optimization over continuous spaces*. — Tech. Rep. No. 95-012, Int. Compt. Sci. Inst., Berkeley, CA.

Tierra Home Page:
www.hip.atr.co.jp/∼ray/tierra/tierra.html.

Tanese R. (1989): *Distributed genetic algorithms*. — Proc. 3rd Int. Conf. *Genetic Algorithms*, Virginia, USA, pp. 434–439.

Töorn A. and Antanas Ž. (1989): *Global Optimization*. — Berlin: Springer.

Tsutsui S. and Fujimoto Y. (1993): *Forking genetic algorithm with blocking and shrinking modes (fGA)*. — Proc. 5th Int. Conf. *Genetic Algorithms*, Urbana-Champaign, USA, pp. 206–213.

Venkateswaran R., Obradović Z. and Raghavendra C.S. (1996): *Cooperative genetic algorithm for optimization problems in distributed computer systems*. — Proc. 2nd Online Workshop *Evolutionary Computation*, Nagoya, Japan, pp. 49–52.

Whitley D., Beveridge R., Graves C. and Mathias K. (1995): *Test driving three 1995 genetic algorithms: New test functions and geometric matching*. — J. Heuristics, Vol. 1, pp. 77–104.

Yi W., Liu Q. and He Y. (2000): *Dynamic distributed genetic algorithms*. — Proc. Congress *Evolutionary Computation*, La Jolla, USA, pp. 1132–1136.

Appendix

Hy4 Configuration

In Table 11, we show the parameter values of each subpopulation for *Hy4*. Each row contains the side configuration concerning the selective pressure ($\eta_{\min}$), the crossover probability (p_c), and the mutation probability (p_m). The heterogeneity of the model can be clearly seen by focusing on each column, where different parameter configurations are applied in each *Hy4* side.

Table 11. Selective pressures ($\eta_{\min}$), crossover rates, and mutation rates for *Hy4* islands.

Side	$\eta_{\min}$	p_c	p_m
Side 1: hE4	0.0	1.0	0.125
Side 1: hE1	0.3	0.7	0.125
Side 1: hE3	0.1	0.9	0.125
Side 1: hE2	0.2	0.8	0.125
Side 2: E4	0.0	0.6	0.125
Side 2: E1	0.3	0.6	0.125
Side 2: E3	0.1	0.6	0.125
Side 2: E2	0.2	0.6	0.125
Side 3: e4	0.8	0.6	0.125
Side 3: e1	0.5	0.6	0.125
Side 3: e3	0.7	0.6	0.125
Side 3: e2	0.6	0.6	0.125
Side 4: he4	0.8	0.6	0.5
Side 4: he1	0.5	0.6	0.2
Side 4: he3	0.7	0.6	0.4
Side 4: he2	0.6	0.6	0.3