Applied Intelligence 12, 163–181 (2000)
© 2000 Kluwer Academic Publishers. Manufactured in The Netherlands.

Influence of the Migration Policy in Parallel Distributed GAs with Structured and Panmictic Populations

ENRIQUE ALBA AND JOSÉ M. TROYA

Dpto. de Lenguajes y Ciencias de la Computación, Univ. de Málaga, Campus de Teatinos (3.2.12), 29071-MÁLAGA, España

eat@lcc.uma.es

Abstract. Parallel genetic algorithms (PGAs) have been traditionally used to overcome the intense use of CPU and memory that serial GAs show in complex problems. Non-parallel GAs can be classified into two classes: panmictic and structured-population algorithms. The difference lies in whether any individual in the population can mate with any other one or not. In this work, they are both considered as two reproductive loop types executed in the islands of a parallel distributed GA. Our aim is to extend the existing studies from more conventional sequential islands to other kinds of evolution. A key issue in such a coarse grain PGA is the migration policy, since it governs the exchange of individuals among the islands. This paper investigates the influence of migration frequency and migrant selection in a ring of islands running either steady-state, generational, or cellular GAs. A diversity analysis is also offered from an entropy point of view. The study uses different problem types, namely easy, deceptive, multimodal, NP-Complete, and epistatic search landscapes in order to provide a wide spectrum of problem difficulties to support the results. Large isolation values and random selection of the migrants are demonstrated as providing a larger probability of success and a smaller number of visited points. Also, interesting observations on the relative performance of the different models are offered, as well as we point out the considerable benefits that can accrue from asynchronous migration.

Keywords: parallel genetic algorithms, complex search spaces, migration policy, entropy, basic island evolution

1. Introduction

Complex optimization tasks need efficient algorithms. Numerous complex learning and search problems are being successfully addressed by evolutionary algorithms (EAs) and particularly by genetic algorithms (GAs) [1]: function optimization, machine learning, combinatorial optimization, constraint satisfaction problems, etc. [2–4]. An EA iteratively improves the sub-optimal solutions contained in a population, thus conducting a search that considers many points of the search space at the same time. For this purpose, the algorithm applies stochastic operators inspired by certain processes found in organic evolution: selection of the fittest, crossover, mutation, etc.

Using a parallel platform to run a parallel GA has the advantage of providing lower run time and enlarging the available memory [5–7]. But, in addition, parallel GAs have an appealing trait in that they often reduce the computational effort to solve the same problem as compared to sequential GAs, even when run in a single processor [8].

This last characteristic makes a difference with respect to other search algorithms, in that PGAs are not simple parallel versions of sequential algorithms. The reason for this can be found in several of the most striking characteristics of a PGA: (1) its decentralized search, which allows *speciation* (different sub-algorithms evolve towards different solutions), (2) the larger diversity levels (many search regions are sought

at the same time) (3) an intense exploration performed by all the *demes* (sub-algorithms), and (4) exploitation inside these demes, i.e., refining the better partial solutions found at any moment.

We will defer a survey of the different kinds of genetic algorithms analyzed in this paper until the next section. For the moment, we will focus our attention on a parallel distributed GA in which many communicating sub-algorithms (islands) cooperate to solve the same problem. In this kind of GA the migration policy (set of rules governing the communication among the islands) biases the search.

Although no formal proof of superiority can be allocated to any migration policy for arbitrary problems [9], some useful research has been conducted on this matter. In this line we further consider a wide variety of problems in order to test our working hypotheses on some of the most well known application domains of GAs. Also, we will check the observations on different parameterizations to avoid biasing the results.

With regard to the migration policy in a distributed GA, the initial work of Tanese [7] on a hypercube (NCUBE/6), and the more recent results of Belding [10] on a KSR2, provided valuable results. These works state that a distributed algorithm can attain important speedup values on problems such as a class of Walsh polynomials and Royal Road functions, respectively. See [5, 11] for additional background.

A relatively long isolated evolution has been a priori suggested to provide good results in [7, 10] whenever a small number of individuals are being migrated. The sparse migration of a small number of strings has the advantage of requiring a low overhead due to the sparse communication among the islands when the distributed GA is run on parallel hardware. Here, we carry out a generalization of the mentioned works since they study multiple parameters at a time, or only a single class of problems is considered. In this work our goal is to check these results on a wide variety of problems and models of GAs in order to better understand the importance of the migration policy.

Therefore, one of our main contributions is to extend the existing analyses to incorporate steady-state and cellular island evolution modes [12, 13], since Tanese and Belding only studied a distributed generational GA. Also, we will consider a wide variety of problems across which the numerical behavior will be studied, and finally, diversity and run time will be discussed to obtain a full picture of the implications of using a given migration policy.

The paper is organized as follows. In Section 2 we characterize parallel GAs and also the migration policy. Sections 3–7 study the influence of the migration policy on easy, deceptive, multimodal, NP-Complete, and epistatic problems, respectively. Afterwards, we analyze the influence of the migration frequency on the computation time (Section 8), and population diversity (Section 9). Finally, the most important conclusions are summarized in Section 10.

2. A Brief Survey of Parallel Genetic Algorithms

In order to offer a brief review of the research with parallel models of GAs we include some information for non-GA researchers in this section. We briefly want to show how coarse grain and fine grain PGAs (cgPGA and fgPGA, respectively) are subclasses of the same kind of parallel GA consisting in a set of communicating sub-algorithms. Although many surveys [2, 3] and studies have suggested an opposite vision/use of these models, they can be studied from a common point of view [14] with potential advantages arising from a unified methodology, just as it is occurring with EAs in general [15].

2.1. *Unified Review of Existing PGAs*

Let us begin with a brief unifying survey of sequential and parallel GAs. A sequential GA (Fig. 1) proceeds in an iterative manner by generating new populations of strings from the old ones. Every string is an encoded (binary, real, ...) version of a tentative solution. An evaluation function associates a fitness measure with every string indicating its suitability as a solution to the problem. The algorithm applies stochastic operators such as selection, crossover, and mutation, on an initially random population in order to compute a new generation of strings [4].

A sequential GA (Fig. 2(a)) applies the genetic operators to the whole population (panmixia). Two popular

```
Initialize population          // With randomly generated solutions
Repeat t = 1, 2, ...                      // Reproductive loop
      Evaluate solutions in the population
      Perform competitive selection
      Apply variation operators          // Crossover, mutation, ...
Until convergence criterion satisfied
```

Figure 1. Pseudocode of a sequential genetic algorithm.

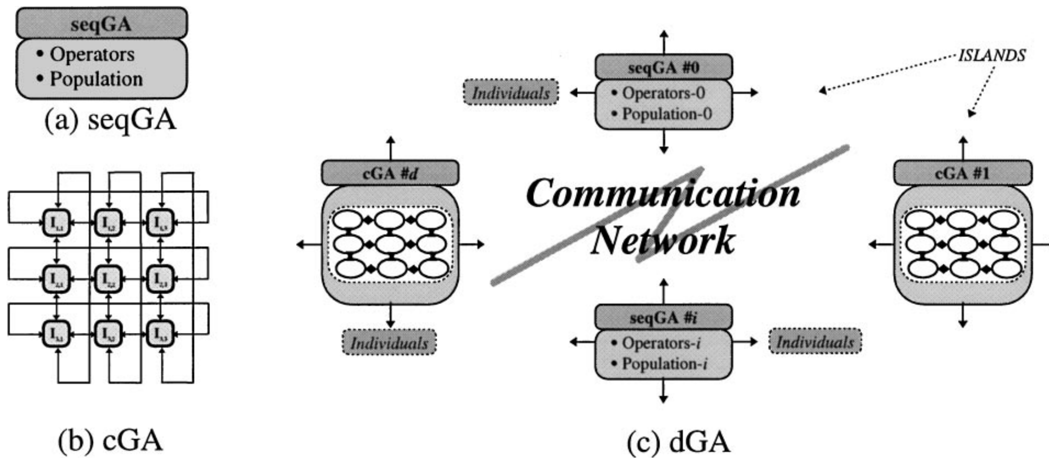


Figure 2. Two basic non-distributed models (sequential-panmictic and cellular) and a possible -merged!- parallel distributed implementation. We call seqGA (a) to a traditionally sequential GA with a steady-state (ssGA) or generational (genGA) evolution. A cellular GA (cGA) uses a given two dimensional spatial distribution for the individuals (b). Finally, a distributed GA (dGA) performs a parallel search (c) by using the mentioned algorithms as the basic reproductive loops in each sub-algorithm (island).

panmictic versions exist: generational and steady-state evolution modes [16]. A generational GA (genGA) creates one new population from the old one, while a steady-state GA (ssGA) only generates one individual (or a few) in each iteration.

In a cellular GA (cGA) [6, 17] the string population is spatially structured with some topology, e.g., a 2D toroidal grid. In a cGA, the interactions of a string with its neighbors (e.g., north, south, east, west) are very frequent. These interactions consist in getting a small pool of $(4+1=5)$ strings to apply the reproductive plan on it (Fig. 2(b)).

A parallel GA is an algorithm of multiple component GAs, regardless of their population structure. Each component (usually a sequential) sub-algorithm includes an additional phase of *communication* with a set of neighboring sub-algorithms. Different parallel algorithms differ in the characteristics of the elementary GAs and in the communication details.

As an example, in distributed GAs (Fig. 2(c)) [7] there exists a small number of islands performing a separate GA, and periodically exchanging strings after a number of isolated steps (*migration frequency*). Normally, the islands use a considerable population size ($\gg 1$), although some works like [7] consider a wide range of population sizes (even as small as two strings per island). Usually, the islands of a coarse grain PGA apply a generational GA, although there are many important exceptions in the literature (such as GENITOR II [18]).

The source and the destination islands for a migrant string are determined depending on the island topology. This topology can be a static ring, hypercube, etc. [2], or a dynamic topology, in which target islands are defined by an arbitrarily complex (time-varying) criterion (e.g., [19]) that is applied in every communication step.

In the cGA model it is usual for every sub-algorithm to wait synchronously for its neighbor structures (suitable for SIMD computers [6], and even implemented in MIMD machines [20]). On the other hand, the distributed GA can either be implemented to have synchronous receptions (wait for incoming migrants) or not, and it is suitable for MIMD computers, e.g., a cluster of workstations. This is an additional advantage of distributed GAs, since clusters are readily available in labs and departments. Therefore, the traditional distribution of sub-populations can be considered as a mechanism capable of enhancing the behavior of any kind of basic GA.

Distributed generational GAs have attracted the greatest attention [5, 7, 10, 11], while some distributed algorithms running steady-state islands exist [13, 18]. However, distributed GAs with cellular islands are relatively new [13, 21, 22]. Here, by using a parallel distributed GA with every island running a cGA we want to obtain the search advantages of a structured-population, as well as the physical-numerical benefits of a parallel distributed GA. Some related works dealing with the migration policy and new models of

evolution for distributed GAs can be found in [23] and [24], respectively.

We should differentiate between the *population structure* (panmictic or decentralized: distributed, cellular, ...), and its *implementation* (hardware type, synchronism, model-to-hardware mapping, ...). We propose a change in the nomenclature, calling coarse and fine grain PGAs *distributed* and *cellular* GAs (dGA and cGA) respectively, since the grain is usually intended to refer to their computation/communication ratio, while the actual differences can be also found in the way in which they both structure their populations.

We assign the name dssGA to a parallel distributed GA running a steady-state algorithm in every island with sparse migrations among islands. Similarly, we define a dcGA to be a distributed algorithm whose islands are executing a cGA, and a dgenGA an algorithm in which the islands are performing a canonical genGA. We implement all the distributed versions in an ATM LAN by mapping each island to a different processor (UltraSparc 1). The islands are configured in a unidirectional ring topology (easy and fast). We mainly focus on the dssGA and dcGA models, and defer the studies with dgenGA until the diversity section. We do this because dgenGA is better known.

While a distributed GA has "large" sub-populations ($\gg 1$) a cGA has typically one single string in every sub-algorithm. For a dGA the sub-algorithms are loosely connected, while for a cGA they are tightly connected. In addition, in a dGA there exist only a few sub-algorithms, while in a cGA there is a large number of them. See the structured-population cube in Fig. 3.

This view of PGAs as a continuum is *directly* supported in only a few implementations of PGA software [21, 25]. However, it is very important since it offers a unifying point of view: we can study the full set of parallel implementations by considering different codings, operators, and communication details.

In fact, this is a natural vision of PGAs that is inspired by the initial work of Holland [26], in which a *granulated adaptive system* is a set of grains with shared structures. Every grain has its own reproductive plan and internal/external representations for its structures (see also [27]). To link our results and algorithms to these works we will develop the survey and technique descriptions with a more formal notation.

The outline of a general PGA is presented in Algorithm 1. It begins by randomly creating a population $P(t=0)$ of μ structures −(strings), each one encoding the p problem variables, usually as a vector over $\mathbb{B}=\{0,1\}$ $(I=\mathbb{B}^{p\cdot l x})$ or $\mathbb{R}(I=\mathbb{R}^p)$. An evaluation function Φ is needed to associate a quality real value with every structure.

The stopping criterion ι of the reproductive loop is to fulfill some condition. The solution is identified as the best structure ever found.

The *selection* s_{Θ_s} operator makes copies of every structure from $P(t)$ to $P'(t)$ attending to the fitness values of the structures (some parameters Θ_s might be required). The typical variation operators in GAs are *crossover* ($\otimes$), and *mutation* (m). They are both stochastic techniques whose behavior is governed by a set of parameters, e.g., application probabilities: $\Theta_c=\{p_c\}$-usually high- and $\Theta_m=\{p_m\}$-usually low-.

Finally, every iteration ends by selecting the μ new individuals for the new population (*replacement policy r*). For this purpose the new pool $P'''(t)$ plus a set Q are considered. The best structure usually survives

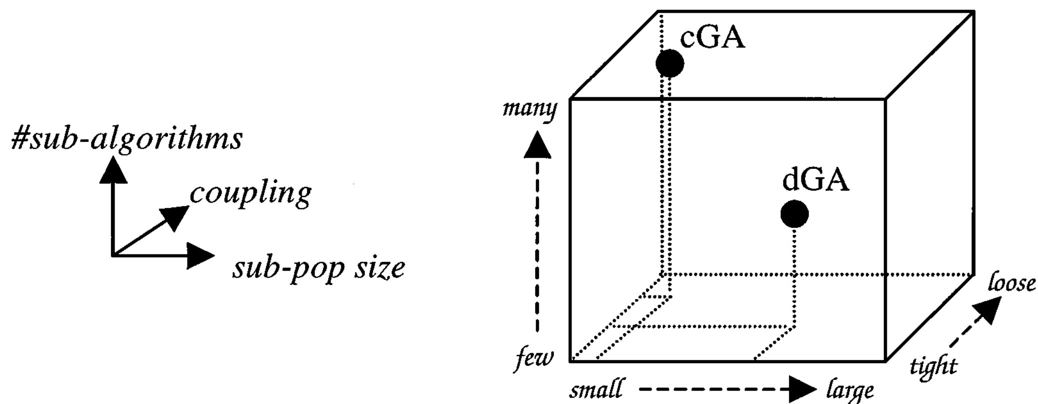


Figure 3. The Structured-Population Genetic Algorithm Cube: cellular GAs (cGA) versus distributed GAs (dGA) and the continuum between them.

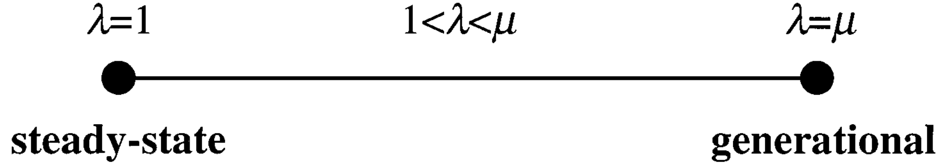


Figure 4. Difference between steady-state and generational GAs. The intermediate region is occupied by the algorithms generating and replacing only a given percentage of the population.

deterministically (*elitism*). Many panmictic variants exist, but the two mentioned are especially popular [16] (Fig. 4): the *generational GA*-genGA- ($\lambda = \mu$, $Q = \emptyset$), and the *steady-state GA* -ssGA- ($\lambda = 1$, $Q = P(t)$).

In a parallel GA there exist the mentioned elementary GAs (grains: Δ_i) working on separate subpopulations $P^i(t)$ concurrently ($\Delta_1 \| \Delta_2 \| \ldots \| \Delta_d$). Each sub-algorithm includes an additional phase of periodic *communication* with a set of neighboring sub-algorithms located on some topology.

This communication usually consists in exchanging a set of individuals or population statistics. All the sub-algorithms perform the same reproductive plan. Otherwise the PGA is heterogeneous [14].

Throughout this paper we deal with MIMD implementations (Fig. 2(c)) of homogeneous dGAs in which migrants are selected *randomly*, and the target island replaces its *worst* string with the incoming one only if it is better (except when other policy is mentioned explicitly). The sub-algorithms are organized in a unidirectional ring.

2.2. Migration Policy in a Parallel Distributed GA

To complete the working principles of these algorithms we now describe their communication phase. The migration policy determines the communication step

ALGORITHM 1: PARALLEL GENETIC ALGORITHM $\Delta_{par} ::= \Delta_1 \| \Delta_2 \| \ldots \| \Delta_d$

Δ_i: $t := 0$;
 initialize: $P(0) := \{\vec{a}_1(0), \ldots, \vec{a}_\mu(0)\} \in I^\mu$;
 evaluate: $P(0) : \{\Phi(\vec{a}_1(0)), \ldots, \Phi(\vec{a}_\mu(0))\}$;
 while not $\iota(P(t))$ **do** //*Reproductive Loop*
  select: $P'(t) := S_{\Theta_s}(P(t))$;
  recombine: $P''(t) := \otimes_{\Theta_c}(P'(t))$;
  mutate: $P'''(t) := m_{\Theta_m}(P''(t))$;
  evaluate: $P'''(t) : \{\Phi(\vec{a}_1'''(0)), \ldots, \Phi(\vec{a}_\lambda'''(0))\}$;
  replace: $P(t+1) := r_{\Theta_r}(P'''(t) \cup Q)$;
  <***communication step***> //*Migration in a distributed GA*
  $t := t + 1$;
 end while

In particular, the steady-state panmictic algorithm we will analyze (Fig. 2(a)) generates one single individual in every iteration. It is inserted back into the population only if it is better (greater fitness than) the worst existing individual.

In all the cGAs we use (Fig. 2(b)) a NEWS neighborhood is defined (**N**orth-**E**ast-**W**est-**S**outh in a toroidal grid [17, 28]) in which overlapping demes of 5 strings (4 + 1) execute the same reproductive plan. In every deme the new string computed after selection, crossover, and mutation replaces the current one only if it is better.

undertaken by the islands in a parallel distributed GA. We define the migration policy as a tuple of five values:

$$\boldsymbol{M} = (m, \zeta, \omega_S, \omega_R, sync) \tag{1}$$

where:

- m: is the number of individuals undergoing migration, $m \in \{0, 1, \ldots, \infty\}$ (migration rate). Alternatively, it could be measured as a sub-population percentage. This value is usually bound by the population size in practice, but, theoretically, nothing

prevents sending many more (copies of) individuals to one or more neighbor islands.

- ζ: is the frequency of migration (in numbers of evaluations), $\zeta \in \{0, 1, \ldots, \infty\}$. With this definition, if the distributed algorithm stops after e evaluations, then any value of migration above this value ($\zeta > e$) means isolated evolution (just like using $\zeta = 0$).
- ω_S: is the policy for selecting migrants. The topology and the migrated individuals are defined by this operator: if two islands do not share any individual they are said to be unconnected (thus defining the topology). This operation takes two islands as arguments, and it also determines the set of shared individuals between any pair of islands in the distributed GA. The migrant selection can be made according to any of the selection operators available in the associated literature (proportional selection, tournament, random, etc.). Also, we could choose between sending out a copy of the selected individual (we do so in this paper) or the individual itself.
- ω_R: is the migration replacement policy, used for integrating an incoming individual in the target subpopulation.
- *sync*: is a flag indicating whether the algorithm is performing regular blocking input/output communications from/to another islands, or whether individuals are integrated in the receiving population whenever they arrive from its neighbors. These two variants can be implemented on a parallel hardware or by using a hand-made scheduler in a uniprocessor machine (*simulating* the parallel operations).

We define the migration frequency in terms of the number of evaluations made in the island, and not in terms of the number of generations, since we are comparing models of different basic steps. A distributed GA with d steady-state islands must take μ/d steps to complete one generation, while the genGA and cGA islands will compute a full new generation of μ individuals in every parallel iteration. Anyway, in all the cases we will study the frequency of migrations in multiples of the global population size: $1 \cdot \mu, 2 \cdot \mu, 4 \cdot \mu, \ldots$ The graphs in this paper show these multiples: 1, 2, 4, ... Hence, ζ depends on the population size, and has no "general recipe" values. When the population is large enough, values between 0 and 1 for ζ could be of some help to complete the analysis.

Formally, a migration operator ω_M is used in the communication phase of a PGA Δ_{par}, indicating how the structures of one island are shared by another island(s). The selection operator $\omega_S(\Delta_i, \Delta_j)$ determines the set of shared structures between neighboring sub-algorithms Δ_i, Δ_j:

$$\omega_{M\Theta_M}(\Delta_j) = \omega_R \circ \omega_S(\Delta_i, \Delta_j) \mid \forall \Delta_i, \Delta_j \in \Delta_{par} \quad (2)$$

The number of migrants is the number of shared structures:

$$m = |\omega_S(\Delta_i, \Delta_j)| \quad (3)$$

and the probability of application of the migration operator is

$$P_M = \frac{1}{\zeta} \quad \zeta > 0 \quad (4)$$

Some work is available on the convenience of using asynchronous communications [13, 20, 29]. This can be achieved by inserting an individual whenever it arrives, thus avoiding blocking every ζ steps, i.e., emissions and receptions of migrants are managed in separate portions of the source code. The set of parameters Θ_M controlling the migration operator ω_M is the migration policy $\boldsymbol{M}$. We will label synchronous results with an **s**, and asynchronous ones with an **a** in the forthcoming graphs.

The reproductive cycle of a parallel distributed GA is then a composition of an island reproductive cycle and a migration operator:

$$\omega_d = \omega_M \circ \omega_{island} \quad (5)$$

In practice, many useful combinations of these techniques are possible, and this is why we set some parameters. We will use a unidirectional ring of islands since it is very easy to implement, and the migration takes place in a constant time (see an interesting discussion on this and other topologies in [30]) which is good for the scalability of the algorithm. Migration rate is set to $m = 1$ while migration frequency will be modified in the experiments, since a correlation between the two seems to exist [10]. The replacement technique in all the experiments eliminates the worst string in the target island only if the incoming one is better, as in [8, 31, 32]. Other works use a random replacement [7] that sometimes allows fitness decrements for escaping from local optima. However, complex problems usually force to preserve at least the best k strings in every island (k-elitism), since losing the best strings cannot be afforded.

In the following four sections we will study the impact of migrant selection and migration frequency on search spaces with different characteristics. In particular, the number of evaluations to obtain a solution, the percentage of hits in the set of independent executions, and the run time will be analyzed.

3. Results with a Traditional Easy Problem

We begin by analyzing the influence of some of the migration parameters in the generalized sphere problem. This is a non-epistatic problem in which the sum of the separated fitness contributions of the variables are maximized (see Eq. (6)).

$$f(x_i \mid_{i=1..n}) = \sum_{i=1}^{n} x_i^2 \quad x_i \in [-5.12, 5.12] \qquad (6)$$

We use an instance of $n = 16$ variables encoded in 32 bits each (SPH16-32, $l = 512$ bits) instead of the usual $n = 3$ problem. Since we are *maximizing*, the problem has 2^{16} possible solutions. Other authors usually minimize this function (there exists a single optimum in **0**). Our version is slightly more difficult since solutions of different genotype could show similar fitnesses, and the algorithm is likely to change many times from one search region to another during one run, thus requiring a longer evolution.

We use $512/n_{proc}$ individuals for the tests, applying fitness proportional selection, double point crossover ($p_c = 1.0$), and bit-flip mutation ($p_m = 1/l$). In the distributed algorithms we send a copy of one random individual to the neighbor population in the ring. Figure 5 plots the average curves (100 runs) of the total number of evaluations for dssGA and dcGA. We study the effects of synchronization (**s**: sync, **a**: async) and the migration frequency (the numeric values associated with the **s** or **a** labels in Fig. 5). The stopping condition is to locate an optimum for this problem.

Three interesting observations can be made from studying Fig. 5. First, a high coupling ($\zeta = 1$) reduces the total effort to locate a solution. This occurs because the panmictic-like evolution of a group of highly coupled islands leads itself to solving this easy problem. The contrary usually holds for more complex problems, as we will see in the next sections.

Second, the larger the number of islands (and processors), the smaller the evaluation effort needed to find a solution (especially for dssGA, see Fig. 5 (left)). This effect is due to the combined work of the ring topology [30], the separate parallel exploration of different zones of the search space provided by the distributed search, and the aggressive exploitation typical of the dssGA model.

Finally, it can also be seen that synchronous and asynchronous versions perform (numerically) very similarly for dssGA and dcGA at any migration frequency. The slight differences are due to the physical parallel checking of the termination condition, and to the fast fitness evaluation of this objective function.

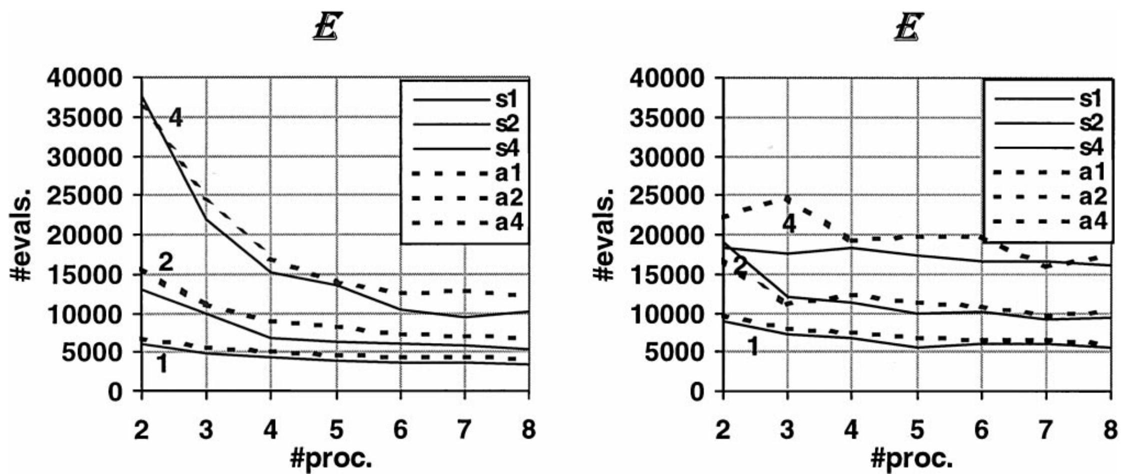


Figure 5. Total number of evaluations versus the number of processors when using dssGA (left) and dcGA (right) for solving SPH16-32. We show the results for three migration frequencies 1, 2, 4 and for synchronous -s- and asynchronous -a- versions.

4. Results with a Deceptive Problem

Deceptive problems are specifically designed to make GAs converge to wrong regions of the search space. These problems are constructed to decorrelate the relationship between the fitness of a string and its genotype (contents).

Since the algorithm works on the genotype, but, at the same time, it is guided by the fitness of the strings (computed after the phenotype), this kind of functions tends to mislead the GA. One of these deceptive problems is the massively multimodal deceptive problem (MMDP) [33] in which a binary string encodes k 6-bit sub-problems (see Fig. 6). The optimum fitness is k. Every sub-problem contributes with a partial fitness depending on its number of 1's (*unitation*). There exist 22^k optima from which only 2^k are global optima. The literature addresses instances for $k = 1, \ldots, 5$, while we will use a considerably more difficult instance of $k = 15$ (strings of $l = 90$ bits).

We study the influence of the migration frequency ζ (0.25, ..., 32.0), and the migrant selection in the ring ω_S: should we send the best or a random string? We plot the number of hits (out of 50 runs) on dssGA and dcGA in Fig. 7 (left). We analyze a configuration of 8 islands with 32 individuals each for all the algorithms (every island in dcGA uses a grid of $2 \times 16 = 32$ individuals). We use proportional selection, two point crossover with $p_c = 1.0$, and bit flip mutation with $p_m = 0.01$.

The results indicate that it is better to migrate a random string since migration of the best string often generates super-individuals which induce premature convergence in the target population. For larger sub-populations this could be a minor problem, but population sizes like ours (in the range [100..500]) are the most frequently used in EAs. A second observation is that loosely coupled islands ($\zeta = 16$ or 32) provide better efficacy, since high isolation promotes hyperplane exploitation in islands, and exploration occurs

BIPOLAR DECEPTION (6 bits)	
#ONES	**sub-function value**
0	1.000000
1	0.000000
2	0.360384
3	0.640576
4	0.360384
5	0.000000
6	1.000000

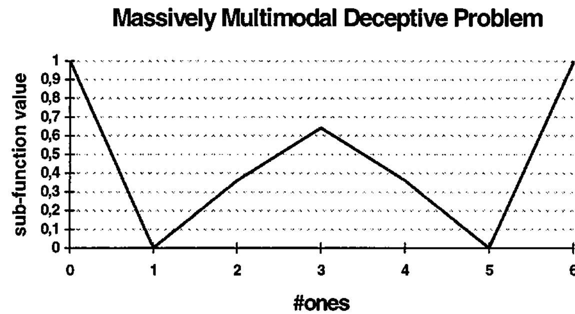


Figure 6. Tabulated (left) and graphic (right) sub-problem contributions for MMDP ($k = 1$).

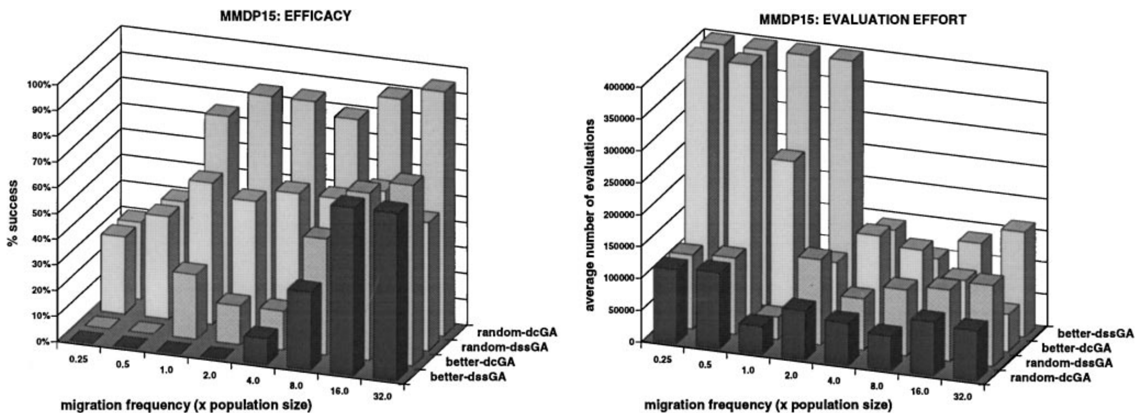


Figure 7. Influence of the migration frequency and migrant selection in MMDP15 with dssGA and dcGA. Success percentage (left) and number of evaluations (right), out of 50 independent runs. The stopping criterion is to obtain a solution or to achieve 400000 evaluations.

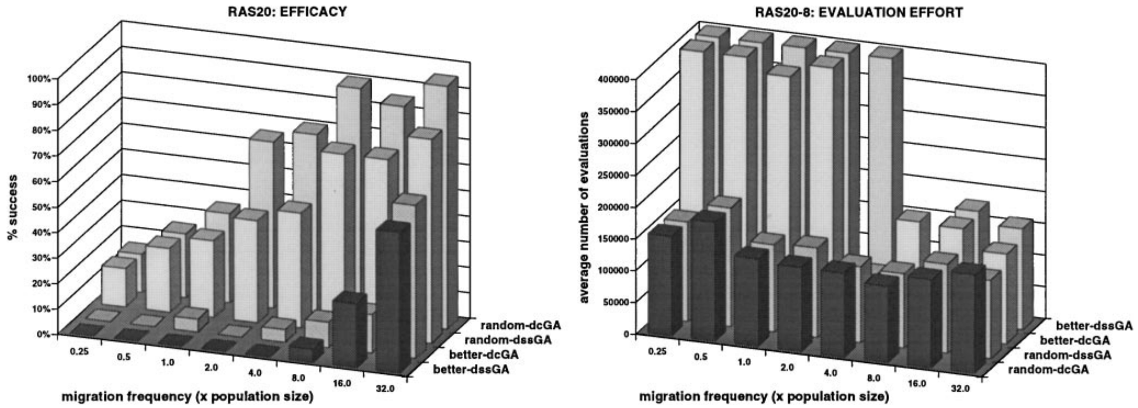


Figure 8. Influence of the migration frequency and migrant selection in RAS20-8 with dssGA and dcGA. Success percentage (left) and number of evaluations (right), out of 50 independent runs. The stopping criterion is to obtain a solution or to achieve 400000 evaluations.

after migrations (punctuated equilibria working principle [34]).

The third observation is that dcGA yields better results than the dssGA in all these experiments with equivalent configurations. In the next section we will often find the same result. We think that this is due to the good exploration capabilities of a cGA, since this algorithm will show to maintain high diversity throughout the search in all the tested problems. Some results support this hypothesis, since ssGA is better for problems where the exploitation component of the algorithm is the main feature needed to locate a solution, e.g., where local tuning is essential (see an example when training a neural network in Section 7).

As a final remark, we have to mention that the results in Fig. 7 (also in Fig. 8) for migration frequencies below 2.0 are wrapped to 400000 evaluations to clarify the graph. All the percentages and values are reported for algorithms reaching a solution of the same quality (this holds throughout the article). In Figs. 7 and 8 we stop the algorithm after performing 400000 evaluations without finding the optimum for frequencies of 2.0 and below. Whether or not these frequent migrations would have led to an optimum, the conclusion is that they performed considerably worse than sparse migrations. For the evaluation effort in Figs. 7 and 8 we average all the runs, considering that unsuccessful ones need 400000 evaluations.

5. Results with a Multimodal Problem

In this section we study the effects of migrant selection and migration frequency on a problem with a large search space and a very large number of local optima that can easily lead to non-optimal solutions. The generalized Rastrigin function (Eq. (7)) is a non-epistatic function representing a typical test for EAs [35]. We use an instance of $n = 20$ variables, each one encoded in 8 bits (RAS20-8, $l = 160$ bits).

$$Ras(x_i\,|_{i=1\ldots n}) = 10 \cdot n + \sum_{i=1}^{n} x_i^2 - 10 \cdot \cos(2 \cdot \pi x_i)$$
$$x_i \in [-5.12, 5.12] \quad (7)$$

We use 8 islands of 22 individuals (grids of $2 \times 11 = 22$ individuals in every island of dcGA), proportional selection, two point crossover with $p_c = 1.0$, and bit-flip mutation with $p_m = 0.01$. We have only changed the population size with respect to the previous section since we are tackling a different problem, and also in order to check whether the achieved observations hold for more reduced populations than the ones tested for the preceding problem.

The results in Fig. 8 confirm the benefits of migrating a random string in that greater efficacy is achieved with moderate effort. Although the numeric effort is very similar for dssGA and dcGA, the number of hits for dcGA is clearly superior. Also, a tight coupling ($\zeta < 16$) is definitely bad for dssGA, although dcGA is capable of a larger number of successful runs even for high interconnection. The experiments with RAS20-8 also corroborate $\zeta = 32$ as a very good migration gap.

6. Results with an NP-Complete Problem

In this section we deal with an optimization task known as the subset sum problem [36]. We have selected this problem because it is an example of the very

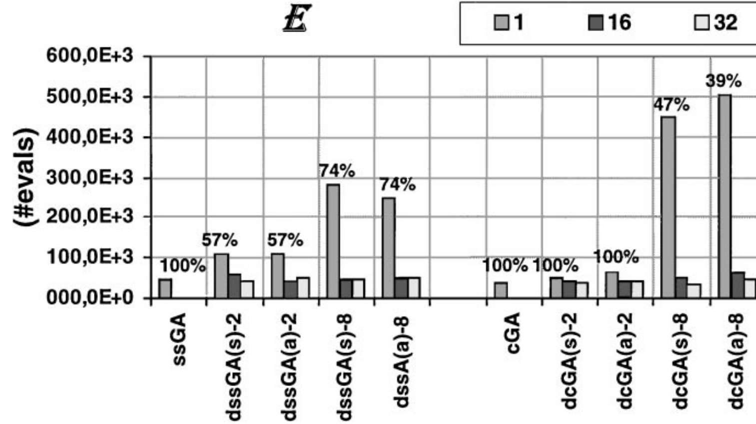


Figure 9. Average number of evaluations with 1, 2, and 8 processors in solving SSS128 with dssGA (left hand of the graph) and dcGA (right hand of the graph). We analyze the results with migration frequencies 1, 16 and 32.

interesting NP-Complete problem class. It consists in finding a subset of values $V \subseteq W$ from a set of integers $W = \{\omega_1, \omega_2, \ldots, \omega_n\}$, such that the subset sum approaches a constant C without exceeding it. We use instances of $n = 128$ (SSS128, $l = 128$ bits) in which the integers ω_i are generated in the range $[0..10^4]$ instead of $[0..10^3]$ in order to obtain a harder problem, as explained in [36].

In order to formulate this as a maximization problem we compute the sum $p(\vec{x}) = \sum_{i=1}^{n} x_i \cdot \omega_i$ for a tentative solution $\vec{x}$, and then use the fitness function:

$$f(\vec{x}) = a \cdot P(\vec{x}) + (1 - a) \cdot max[C - 0.1 \cdot P(\vec{x}), 0] \quad (8)$$

where $a = 1$ when $\vec{x}$ is admissible, i.e., when $C - P(\vec{x}) \geq 0$, and $a = 0$ otherwise. Solutions exceeding the constant C are penalized.

The experiments (Fig. 9) plot dssGA and dcGA over $n_{proc} = 1, 2, 8$ processors for three values of the migration frequency (1, 16, 32) and a random selection of migrants.

We performed 100 independent runs and used $512/n_{proc}$ individuals. The dcGA uses either 2 islands with a 2×128 grid, or 8 islands with a 2×32 grid. We always apply proportional selection, two point crossover with $p_c = 1.0$, and bit-flip mutation with $p_m = 1/l$.

When a random string migrates with migration gap 1, the number of visited points is bad (large) for 2 and 8 processors, even precluding finding a solution. We plot the percentage of success on the top of the bars only for frequency $\zeta = 1$.

Migration frequencies of 16 and 32 provided much better numeric efficiency and 100% efficacy in all the experiments. Also, better resistance of dcGA to bad migration frequencies ($\zeta = 1$) for a small number of large islands and of dssGA for a larger number of small islands is detected. For example, for migration frequency $\zeta = 1$, when using 2 processors, dssGA had 57% success while dcGA had 100% and it needed a smaller number of evaluations. On the other hand, when using 8 processors, dssGA had a 74% success rate while dcGA had 47%–39% for sync-async versions, respectively (always with $\zeta = 1$).

These results can be explained as follows. The frequent migration has a larger impact on dssGA when few populations are being used, since its numerical behavior is more similar to that of a panmictic ssGA: it has more difficulties in solving a complex problem if a panmictic-like evolution is used. On the other hand, the negative impact of higher migration frequency is more important for dcGA as the total grid is split into smaller sized islands: two islands of $2 \times 128 = 256$ individuals had better resistance to frequency $\zeta = 1$ than eight islands of $2 \times 32 = 64$ individuals each. The reason is that structuring the population is clearly worthwhile only for medium/large population sizes.

7. Results with an Epistatic Problem

Epistasis is a fundamental concept in problem complexity [37]. It defines the degree of parameter interdependency present in a problem. Real-life and difficult domains are epistatic. A clear example of an epistatic

problem can be found in training a neural network (NN), since the correlation among the weights is very high [32].

We use a PGA for learning the set of weights that make a multilayer perceptron classify a set of patterns correctly [38]. This is an example of integration of technologies in the field of *computational intelligence* that has provided valuable results in escaping from local optima, since the weight search space is multimodal. In addition, other different applications are possible, such as using GAs for selecting the most representative pattern subset from a large pattern set in order to reduce the training time of a neural network. In this paper we only deal with training a NN by genetic means.

The first NN we consider computes the parity of 4 binary inputs [39] and it uses three layers of 4-4-1 neurons. The neurons have a *binary* activation function. Every string contains $l = 4 \cdot 4 + 4 \cdot 1 + 4 + 1 = 25$ real-encoded variables (weights plus biases). The fitness function computes the absolute error between the expected and the actual output over the 16 training patterns. The result is subtracted from the maximum possible error to have a maximization function.

The second NN is trained to predict the level of urban traffic in a road [40]. The perceptron has three layers of 3-30-1 neurons with a *sigmoid* activation function. For this problem we present results by encoding each weight as a real number or in a gene of 8 bits. This yields strings of $l = 151$ real values or $l = 1208$ bit length, respectively. Every function evaluation computes the error along a set of 41 patterns.

The sigmoid neurons (continuous neuron outputs), and the pattern set make this problem hard for a canonical GA without using any specialized operator and/or representation. Also, the long length of the strings is an additional problem for the medium-size populations we *want* to use.

For all these experiments we averaged the results over 100 independent runs and use $512/n_{proc}$ individuals (dcGA uses 8 islands of 2×32 strings), proportional and random selection for every pair of parents, uniform crossover [41] with $p_c = 1.0$, and additive float mutation with $p_m = 0.01$. All the algorithms have to show the same final solution quality (a mean error below 10^{-3}), and this is used as the termination criterion.

The numeric results in Fig. 10 (left) indicate that $\zeta = 16$ and $\zeta = 32$ are the best isolation times for the "parity4" NN (just like for the previous problems). Concerning the "traffic" network (Fig. 10 (right)), all the isolation times studied needed very similar numbers of evaluations. This occurs because the main difficulty with this network is in refining the final values of the genomes, a widely known weak characteristic of GAs when no specialized operators are used. Almost all the search time is spent in refining the weights in the strings of the algorithm. Also notice that the number of evaluations that the different models need to solve these problems ($\sim 10^6$) is really large when compared to other problems.

On average, dcGA has the advantage when solving the "parity4" NN (Fig. 10 (left)). However, the contrary is true for the traffic NN (Fig. 10 (right)), with the exception of the slightly greater numerical

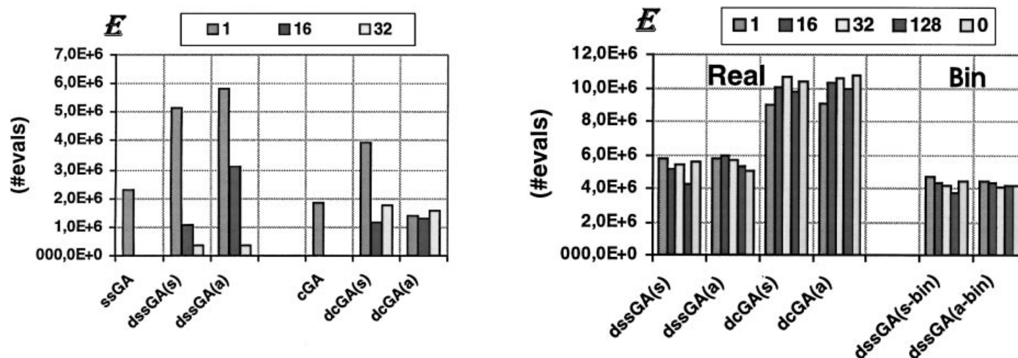


Figure 10. Number of evaluations training "parity4" (left) and "traffic" (right) with several different migration frequencies (1-16-32 and 1-16-32-128-0). We plot results with 1 and 8 processors for "parity4", and 8 processors for "traffic". Besides that, for the "traffic" NN training problem we compare the number of evaluations needed to solve the problem when the weights are encoded in floating point numbers versus a binary representation for them.

effort of synchronous dcGAs using migration frequencies $\zeta = 16$ and $\zeta = 32$.

This supports the hypothesis of the larger exploration/exploitation canonical characteristics of dcGA/dssGA, respectively. This means that, since the "parity4" NN has only a few different fitness values, it requires high hyper-plane diversity and recombination (exploration) to find a solution since the fitness values do not provide much information to the algorithm. This necessity of exploration is easily achieved thanks to the local work in every neighborhood of the cGA and dcGA. On the other hand, for the "traffic" problem, all the algorithms *quickly* found the region of the solution when a distributed model is being used, but they need a very large number of evaluations to tune the real-values included in the strings, since the fitness landscape is continuous. This latter is best achieved in the ssGA and dssGA models since their selection pressure is much stronger than that of the cGA and dcGA versions.

One of the main problems found in other similar works [32] is the loss of diversity. We have no such problems thanks to our floating-point representation, relatively high mutation (no random search is introduced since a bad string never replaces a better one in all the models tested), and good diversity (see Section 9). The analysis of the binary encoding will be discussed in the next section in the light of results on run time.

8. Influence of the Migration Frequency on Run Time

Since the migration policy influences the number of visited points, it also modifies the search time, usually reducing it [7, 10]. For the tests in this section we are using the same problems and parameters as for the numerical results in the last two sections. We measure the total execution time in seconds in all the time graphs (computation plus communications). The time has been measured in a cluster of SUN UltraSparc 1 workstations running Solaris v2.5 and linked by an ATM LAN. Again, in all cases, the termination criterion is to find a solution to the problem studied.

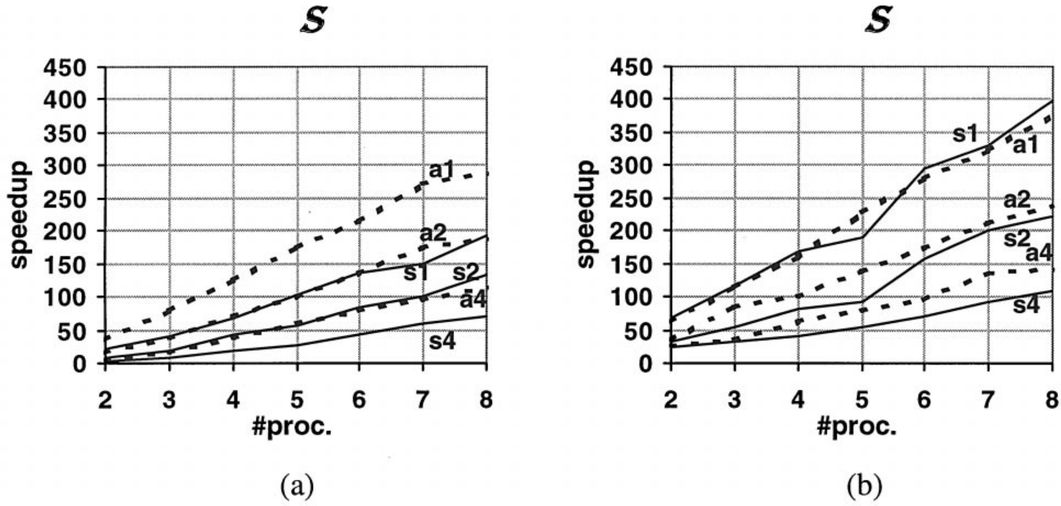
Figure 11 shows the results when solving SPH16-32 with dssGA (a) and dcGA (b). Superlinear speedups are obtained for all the models since the parallel algorithms are much faster than the sequential ones: ssGA and cGA. Notice in the table of Fig. 11(c) how an ssGA needs 548.44 seconds to find a solution, while a distributed synchronous dssGA with 8 islands and a migration frequency of $\zeta = 1$ locates a solution in only 2.84 seconds. Asynchronous versions are especially consistent, whereas the speedup of the synchronous versions is more dependent on the migration frequency. For the SPH16-32 problem, the speedup of the dcGA models is comparatively better than the speedup of the dssGA models.

In Fig. 12 we plot similar reductions in the run time when using distributed versions of ssGA and cGA for the subset sum problem. There is a clear general trend to decrease the search time as the number of processors increases. However, having a large isolation step is fundamental (Fig. 12 plots three values of isolation: 1, 16 and 32); in some cases a tight coupling ($\zeta = 1$) could prevent the algorithm from presenting linear speedup. In all these cases the asynchronous algorithms outperformed their synchronous versions, as suggested for other parallel models and problems [19, 29].

Long binary representations for complex problems like training an artificial neural network (Fig. 13) need a larger computation time as is also shown in [42]. However, they usually ease the search in that performing binary exchanges among strings is more easily managed than crossing over real numbers in the algorithm with a traditional uniform crossover. Besides this, consider the lower number of evaluations in Fig. 10 (right) needed by the binary (discretized) representation used in dssGA.

There exist many ways in which we could have improved the results shown, namely (1) reducing the crossover probability, (2) using a different crossover operator (in fact, uniform crossover is not theoretically well suited to such a highly epistatic problem), or (3) even changing the encoding. However, we do not want to make these choices in order to avoid biasing the results just to obtain "pretty" performance graphs.

High coupling induces excessive synchronization and larger execution times in almost all the experiments. Conclusions are somewhat different for dssGA and dcGA, since the grain of their basic step is different: a concatenation of μ steps in dssGA is slightly slower than a single step in dcGA. This effect is almost unnoticeable in runs requiring a few hundred thousand evaluations, but it provokes a noticeable difference in the run time when long executions are needed (millions of distributed evaluations). Although it is of interest, this result can not be generalized since it depends on whether the population needs to be sorted or not in every step, the time to compute fitness, etc. For example, using ranking in a panmictic GA of 512 individuals is rather slower than using it in a cellular GA of 512



algs.	ζ	ssGA	2	3	4	5	6	7	8
dssGA	1	548.44	26.54	13.90	7.96	5.38	4.02	3.66	2.84
	2	548.44	58.72	30.34	12.80	9.58	6.62	5.42	4.14
	4	548.44	171.52	72.84	29.72	20.48	12.52	9.10	7.88
		cGA	**2**	**3**	**4**	**5**	**6**	**7**	**8**
dcGA	1	989.70	14.32	8.42	5.86	5.22	3.34	3.00	2.48
	2	989.70	31.30	18.40	12.24	10.86	6.30	4.92	4.44
	4	989.70	42.50	30.82	24.98	18.62	13.92	10.60	9.04

(c)

Figure 11. Speedup solving SPH16-32 with dssGA (a) and dcGA (b). The execution times for the synchronous versions (c) show that large differences can appear when comparing ssGA/cGA with dssGA/dcGA with any of the tested migration frequencies: 1, 2 and 4, and for any number of processors (from 1 to 8).

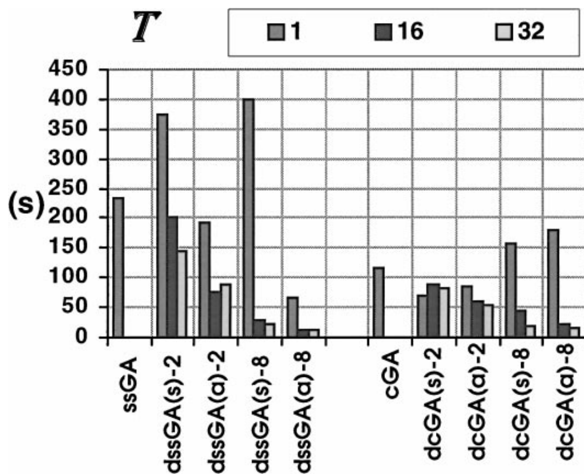


Figure 12. Average execution time in solving SSS128 with dssGA and dcGA (1, 2, and 8 processors) with migration frequencies 1, 16 and 32.

individuals, since in the last case the rank is locally and quickly applied to small groups of 5 strings.

In Fig. 14 (right) we can appreciate that linear and even super-linear speedup values can be attained in practice when the stopping criterion is to obtain a solution of the same quality even for complex problems. This requirement has been imposed lately in various works [29, 30] for ensuring correctness and fairness in the speedup and time studies comparing sequential and parallel GAs. Super-linear speedups are relatively frequent in these stochastic parallel algorithms [5, 10].

Some of the speedup values are surprisingly large, despite comparing algorithms that find a solution of the same quality. The basic reason is that we are comparing the performance of one population (structured or not) against a distributed population partitioned in a distributed island GA. These two algorithms behave

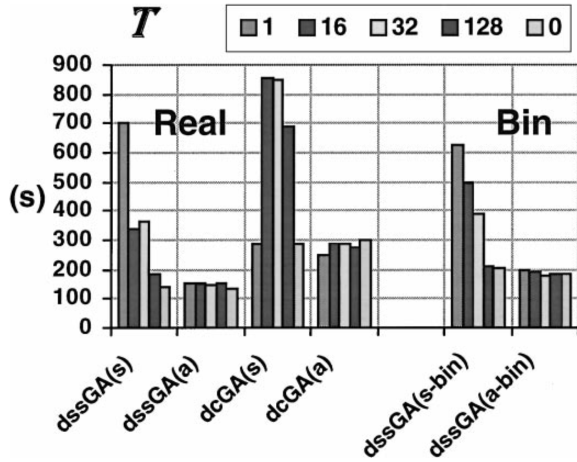


Figure 13. Average execution time in training "traffic" with 8 processors with dssGA and dcGA. Floating-point and binary representations are shown in the graph, and many migration frequencies have been tested (1-16-32-128-0).

differently, thus explaining our results, as well as many other results found in this field.

The natural step after having used the "correct" stopping criterion for a fair comparison is to present speedup values of the same dssGA/dcGA running over 1, 2, ... processors. This would have worked out more reasonable speedup curves. This idea is an ongoing line of research for the authors, and preliminary results show almost linear speedup values, i.e., a traditional result in parallel algorithms when the parallelization scheme is good (but still with sparse moderate super-linear speedup lines).

9. Influence of Migration Frequency on Diversity

A key issue in distributed GAs is that structuring their population induces speciation [7, 10]. This means that different islands can evolve to different solutions. In general, a PGA is a widely used method to sustain diversity [2, 3, 43], thus giving the algorithm more chances to explore new promising zones, and to improve schema processing [19]. But for the same number of individuals as a sequential GA, the distributed islands can show quick convergence due to a reduction in diversity (see Fig. 15).

Figure 15 shows the mean population entropy $\boldsymbol{H}$ (see Eq. (9), in which P_0^i is the proportion of 0's at position $i : 1..l$, and P_1^i is the number of 1's) of a population of $512/n_{proc}$ individuals when solving the generalized sphere problem (see Section 3). Every line is the average of 20 independent runs to provide meaningful results.

$$\boldsymbol{H}[P(t)] = -\frac{1}{l} \cdot \sum_{i=1}^{l} \left(P_0^i \cdot \log_2 P_0^i + P_1^i \cdot \log_2 P_1^i\right) \tag{9}$$

Figure 15(a) plots decreasing diversity curves as the algorithm proceeds for two cellular GAs, one generational, and one steady-state GA. We plot two separate grid cGA shapes (almost square 22×23, and thin 4×128) since the shape seems to be related to the selection pressure, diversity, and exploration capabilities of the cGA [12, 28].

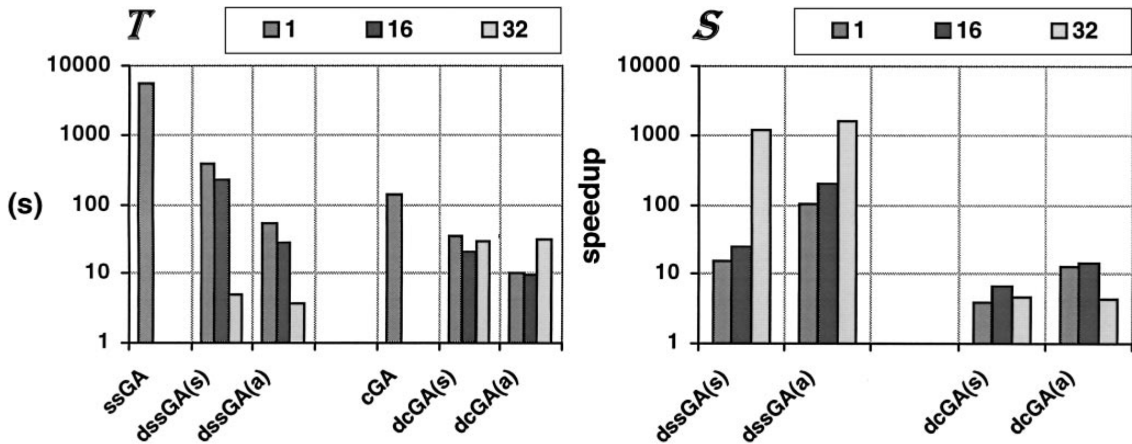


Figure 14. Average execution time in training "parity4" with 1 and 8 processors (left), and speedup with 8 processors (right) with dssGA and dcGA. Three migration frequencies have been analyzed: 1, 16 and 32.

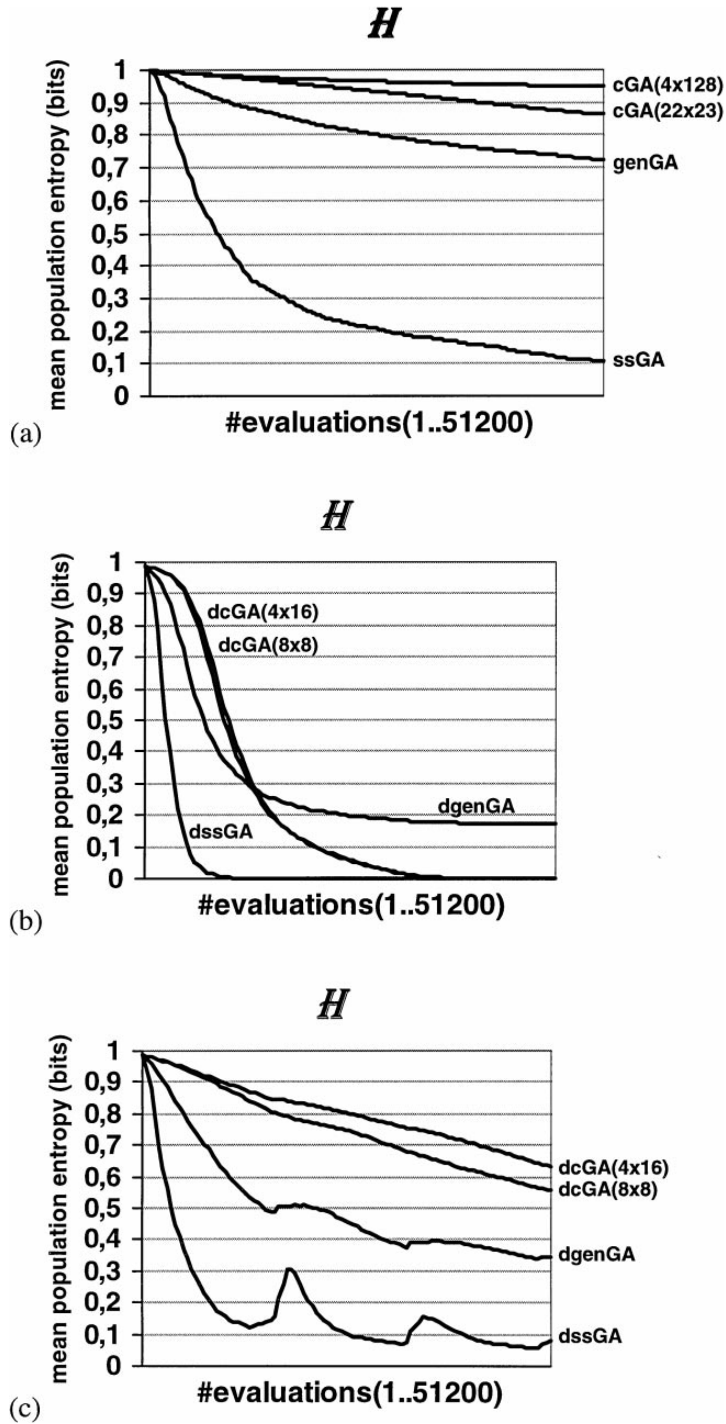


Figure 15. Population mean entropy (in bits) in solving SPH16-32 with non-distributed (a) and distributed models: frequency 1 (b) and 32 (c). We use $512/n_{proc}$ individuals, proportional selection, two point crossover with $p_c = 1.0$, and bit-flip mutation with probability $p_m = 1/l$.

The distributed versions (Fig. 15(b)) show severe entropy drops since the 512 string population is partitioned into 8 islands of 64 strings each. However, the migration frequency has a major impact on the overall quantity of information processed by the algorithm. For a migration frequency of 32 (Fig. 15(c)) the entropy is much higher than for 1 (Fig. 15(b)). Note, for example, the small diversity peaks for dssGA in Fig. 15(c) when migration occurs: loosely coupled islands favor diversity.

The SPH16-32 problem is easy and diversity is quickly lost after a solution is found. For more difficult problems like the subset sum it is normal to have a spectrum of higher diversity. If we compare Fig. 16(a)

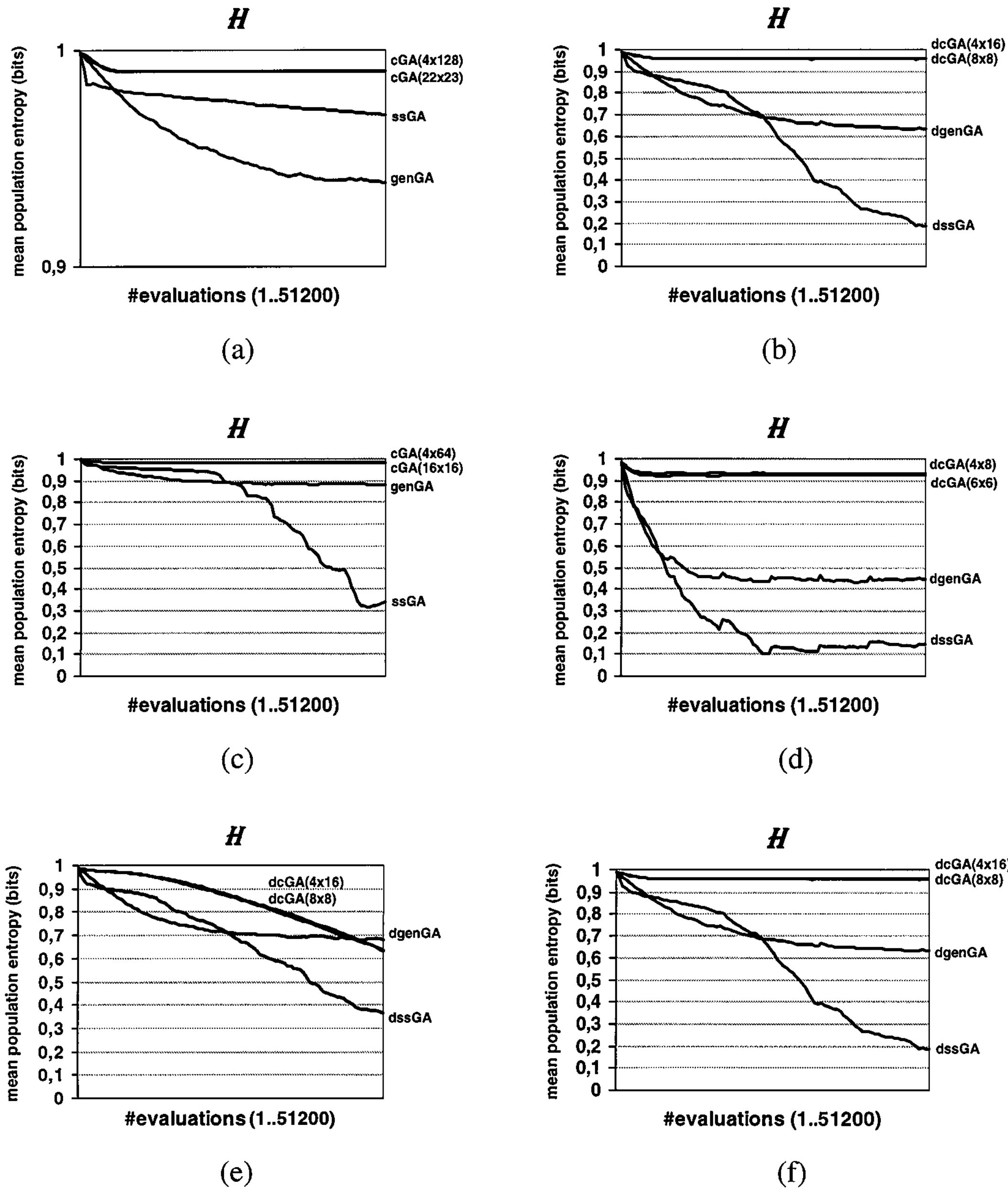


Figure 16. Population mean entropy (in bits) in solving SSS128 with non-distributed (a) and distributed synchronous models with frequency 32 (b). Reducing the number of individuals from $512/n_{proc}$ (first row) to $256/n_{proc}$ also reduces the diversity for non-distributed (c) and distributed models of equal frequency (d). When using 512 individuals again, a high coupling with frequency 1 (e) reduces diversity, while executing the asynchronous versions with frequency 32 (as in b) does not alter the entropy spectrum (f).

and (b) we will notice how diversity considerably decreases for dssGA and dgenGA with 8 islands, while dcGA maintains a very high diversity (consider the numeric labels of the Y axis in Fig. 16(a)). Even dssGA provides a surprisingly high level of diversity. This explains its successful application to this and other domains [12, 31, 32], although it should converge much faster than the other models from a theoretical point of view.

In Fig. 16(c) and (d) we plot the effects of reducing the global population from 512 (first row) to 256 for the non-distributed and distributed models. Of course, diversity decreases quicker, although dcGA still maintains large the allele diversity. This is due to its lower decentralized selection pressure [28] when compared to the other models. This characteristic makes them very suitable for problems where local search is added into the GA or for non-stationary objective functions, since local search often provokes global diversity decrements, and since dynamically changing functions need to preserve diversity to quickly adapt the population to find the new optimal value.

Figure 16(e) plots a serious reduction in diversity for dcGA when the migration frequency is modified from $\zeta = 32$ (loosely coupled) to $\zeta = 1$ (highly coupled). The diversity in dssGA and dgenGA is not reduced, because, while dcGA effectively finds a solution and propagates it quickly, dssGA and dgenGA keep wandering along the search space without working out a solution (thus presenting high diversity but useless search).

Finally, in Fig. 16(f) we can see that executing parallel asynchronous versions of the synchronous algorithms in Fig. 16(b) does not change the diversity spectrum. In fact, the synchronous and asynchronous versions are running the same algorithm on a cluster of workstations of the same type. This provokes differences mainly in the execution time (not in the numerical behavior), since small time derivations are compensated on average, with the added advantage that asynchronous islands are faster since they do not wait for each other at every migration step.

10. Conclusions

In this paper we have confirmed the importance of the migration policy in parallel distributed genetic algorithms, not only for the traditional model of island evolution (a generational GA), but also for steady-state and cellular nodal GAs. PGAs almost always outperformed sequential GAs. They provided shorter execution times and visited a smaller number of points for equivalent parameterizations. However, frequent island interactions can lead to parallel distributed GAs which perform worse than the non-distributed ones, making migration policy an important design factor.

The importance of using considerable isolation times among the islands and the benefits of running asynchronous versions are supported by our results, and indirectly by other works [23, 29]. Migration of a random string prevents the “conquest” effect in the target island for small or medium sized sub-populations.

Using problem-knowledge operators could lead to different results. However, the canonical revision of these techniques is important for their future extensions and combinations with other techniques by providing baselines for comparisons.

We have obtained preliminary results with similar GAs in other domains like TSP and numerical optimization problems that support the benefits of using large isolation steps. “Considerable isolation” has been quantified in this paper as a factor (16 and 32) of the entire population size. These values have been computed to provoke many migrations during the search. If the search uses populations as large as 10^4 individuals our results are thought to hold since the search would need a similarly very large number of evaluations. Very large populations needing a small number of evaluations to find a solution (e.g., of the same magnitude) need a detailed and special study.

Although large isolation gaps and random emigrants cannot be theoretically proven to be the best choices for an arbitrary problem, the heterogeneity of our test suite and their considerable complexity provide strong empirical support to this hypothesis.

Diversity is usually related to exploration and probability of success in the literature. The distributed models can be tuned to provide any desired level of diversity. In particular, the dcGA model is very flexible since the grid shape (or the neighborhood) can be changed to modify selection pressure. This source of flexibility (changing the shape of the 2D grid) is only present in a cGA. The selection operator used in all the tested algorithms is another *different* (complementary) source of exploitation, that can be used to provide an additional level of selection pressure. This observation does not restrict the ability of these or other algorithms to change their selection pressure. It only states the additional source of flexibility coming from using cGA or dcGA.

The impact of the migration policy is still an open question depending on the island reproductive plan and

its relationship to the fitness landscape: do parallel distributed GAs do better with high coupling for easy problems, and are ssGA/cGA more appropriate for exploitation/exploration, respectively? In the light of our heterogeneous (but limited) test suite, and with the same conditions and operators for these two models, the answer in both cases is "yes", although we recognize that more research is needed on this matter.

References

1. T. Bäck, D. Fogel, and Z. Michalewicz (eds.), *Handbook of Evolutionary Computation*, Oxford University Press, 1997.
2. E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs Prallèles, Reseaux et Systems Repartis*, vol. 10, no. 2, pp. 141–171, 1998.
3. A. Chipperfield and P. Fleming, "Parallel genetic algorithms," in *Parallel and Distributed Computing Handbook*, edited by A.Y.H. Zomaya, MacGraw-Hill, pp. 1118–1143, 1996.
4. D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.
5. R. Shonkwiler, "Parallel genetic algorithms," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, edited by S. Forrest, Morgan Kaufmann, San Mateo, CA, 1993, pp. 199–205.
6. P. Spiessens and B. Manderick, "A massively parallel genetic algorithm," in *Proceedings, of the Fourth International Conference on Genetic Algorithms*, edited by R.K. Belew, and L.B. Booker, Morgan Kaufmann, San Mateo, CA, 1991, pp. 279–286.
7. R. Tanese, "Distributed genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, edited by J.D. Schaffer, Morgan Kaufmann, San Mateo, CA, 1989, pp. 434–439.
8. V.S. Gordon and D. Whitley, "Serial and parallel genetic algorithms as function optimizers," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, edited by S. Forrest, Morgan Kaufmann, San Mateo, CA, 1993, pp. 177–183.
9. D.H. Wolpert and W.G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, 1997, pp. 67–82.
10. T.C. Belding, "The distributed genetic algorithm revisited," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, edited by L.J. Eshelman, Morgan Kaufmann, San Francisco, CA, 1995, pp. 114–121.
11. C.C. Pettey and M.R. Leuze, "A theoretical investigation of a parallel genetic algorithm," in *Proceedings of the Third International Conference on Genetic Algorithms*, edited by J.D. Schaffer, Morgan Kaufmann, San Mateo, CA, 1989, pp. 398–405.
12. C. Cotta, E. Alba, and J.M. Troya, "A study on the robustness of parallel genetic algorithms," *Inteligencia Artificial*, vol. 5, no. V/98, pp. 6–13, 1998 (in Spanish).
13. E. Alba and J.M. Troya, "An analysis of synchronous and asynchronous parallel distributed genetic algorithms with structured and panmictic islands," in *Parallel and Distributed Processing*, edited by J. Rolim et al., Springer-Verlag, pp. 248–256, 1999. Lecture Notes in Computer Science 1586.
14. E. Alba and J.M. Troya, "A survey of parallel distributed genetic algorithms," *Complexity*, vol. 4, no. 4, pp. 31–52, 1999.
15. T. Bäck, *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, 1996.
16. G. Syswerda, "A study of reproduction in generational and steady-state genetic algorithms," in *Foundations of GAs* 1, edited by G. Rawlins, Morgan Kaufmann, pp. 94–101, 1991.
17. D. Whitley, "Cellular genetic algorithms," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, edited by S. Forrest, Morgan Kaufmann, San Mateo, CA, 1993, p. 658.
18. D. Whitley and T. Starkweather, "GENITOR II: A distributed genetic algorithm," *J. Expt. Theor. Artificial Intelligence*, vol. 2, pp. 189–214, 1990.
19. M. Munetomo, Y. Takai, and Y. Sato, "An efficient migration scheme for subpopulation-based asynchronously parallel genetic algorithms," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, edited by S. Forrest, Morgan Kaufmann, San Mateo, CA, 1993, p. 649.
20. T. Maruyama, T. Hirose, and A. Konagaya, "A fine-grained parallel genetic algorithm for distributed parallel systems," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, edited by S. Forrest, Morgan Kaufmann, San Mateo, CA, 1993, pp.184–190.
21. N.J. Radcliffe, P.D. Surry, "The reproductive plan language RPL2: Motivation, architecture and applications," in *Genetic Algorithms in Optimisation, Simulation and Modelling*, edited by J. Stender, E. Hillebrand, and J. Kingdon, IOS Press, 1994.
22. F. Gruau, "Neural networks synthesis using cellular encoding and the genetic algorithm," Ph.D. Thesis. Univ. Claude Bernard-Lyon I, 1994.
23. J.G. Maresky, "On efficient communication in distributed genetic algorithms," Master Thesis Dissertation, Institute of Computer Science, The Hebrew Univ. of Jerusalem, 1994.
24. S.H. Lin, E.D. Goodman, and W.F. Punch, "Investigating parallel genetic algorithms on job shop scheduling problems," in *Proceedings of the Sixth International Conference on Evolutionary Programming*, edited by P. Angeline, R. Reynolds, J. McDonnell, and R. Eberhart, Springer Verlag, Berlin, 1997, pp. 383–393.
25. E. Alba, "*Analysis and design of parallel distributed genetic algorithms*," Ph.D. Dissertation, University of Málaga, Spain, March, 1999 (in Spanish).
26. J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.
27. J.M. Daida, S.J. Ross, and B.C. Hannan, "Biological symbiosis as a metaphor for computational hybridization," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, edited by L.J. Eshelman, Morgan Kaufmann, San Francisco, CA., 1995, pp. 328–335.
28. J. Sarma and K. DeJong, "An analysis of local selection algorithms in a spatially structured evolutionary algorithm," in *Proceedings of the Seventh International Conference on GAs*, edited by T. Bäck, Morgan Kaufmann, San Francisco, CA, 1997, pp. 181–186.
29. W.E. Hart, S. Baden, R.K. Belew, and S. Kohn, "Analysis of the numerical effects of parallelism on a parallel genetic algorithm," in *Proc. of the Workshop on Solving Combinatorial Optimization Problems in Parallel*, edited by IEEE, CD-ROM IPPS97, 1997.
30. E. Cantú-Paz and D.E. Goldberg, "Predicting speedups of idealized bounding cases of parallel genetic algorithms," in *Proceedings of the Seventh International Conference on GAs*, edited

by T. Bäck, Morgan Kaufmann, San Francisco, CA, 1997, pp. 113–120.
31. E. Alba, J.F. Aldana, and J.M. Troya, "Genetic algorithms as heuristics for optimizing ANN design," in *Artificial Neural Nets and Genetic Algorithms*, edited by R.F. Albrecht, C.R. Reeves, N.C. Steele, Springer-Verlag, pp. 683–690, 1993.
32. D. Whitley and T. Hanson, "Optimizing neural networks using faster, more accurate genetic search," in *Proceedings of the Third International Conferenceon Genetic Algorithms*, edited by J.D. Schaffer, Morgan Kaufmann, San Mateo, CA, 1989, pp. 391–396.
33. D.E. Goldberg, K. Deb, and J. Horn, "Massively multimodality, deception and genetic algorithms," in *Parallel Problem Solving from Nature 2*, edited by R. Männer and B. Manderick, North-Holland, pp. 37–46, 1992.
34. J.P. Cohoon, S.U. Hegde, W.N. Martin, and D. Richards, "Punctuated equilibria: A parallel genetic algorithm," in *Proceedings of the Second International Conference on GAs*, edited by J.J. Grefenstette, Lawrence Erlbaum Associates, 1987, pp. 148–154.
35. J.M. Yang, J.T. Horng, and C.Y. Kao, "A continuous genetic algorithm for global optimization," in *Proceedings of the Seventh International Conference on Genetic Algorithms*, edited by T. Bäck, Morgan Kaufmann, San Francisco, CA, 1997, pp. 230–237.
36. M. Jelasity, "A wave analysis of the subset sum problem," in *Proceedings of the Seventh International Conference on Genetic Algorithms*, edited by T. Bäck, Morgan Kaufmann, San Francisco, CA, 1997, pp. 89–96.
37. B. Naudts, D. Suys, and A. Verschoren, "Epistasis as a basic concept in formal landscape analysis," in *Proceedings of the Seventh International Conference on GAs*, edited by T. Bäck, Morgan Kaufmann, San Francisco, CA, 1997, pp. 65–72.
38. D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Computing*, vol. 14, pp. 347–361, 1990.
39. S.G. Romaniuk, "Evolutionary growth perceptrons," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, edited by S. Forrest, Morgan Kaufmann, San Mateo, CA, 1993, pp. 334–341.
40. G. Cammarata, S. Cavalieri, A. Fichera, and L. Marletta, "Noise prediction in urban traffic by a neural approach," in *Proc. of the International Workshop on Artificial Neural Networks*, edited by J. Mira, J. Cabestany, and A. Prieto, Springer-Verlag, 1993, pp. 611–619.
41. G. Syswerda, "Uniform crossover in genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, edited by J.D. Schaffer, Morgan Kaufmann, San Mateo, CA, 1989, pp. 2–9.
42. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 1992.
43. D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, pp. 65–85, 1994.