A USEFUL REVIEW ON COARSE GRAIN PARALLEL GENETIC ALGORITHMS

Enrique Alba
Dpto. de Lenguajes y CC.CC.
Universidad de Málaga
Campus de Teatinos (2.2.A.6)
29071-Málaga (ESPAÑA)
eat@lcc.uma.es
http://www.lcc.uma.es/personal/alba/alba.html

José M. Troya
Dpto. de Lenguajes y CC.CC.
Universidad de Málaga
Campus de Teatinos (2.2.A.13)
29071-Málaga (ESPAÑA)
troya@lcc.uma.es
http://www.lcc.uma.es/personal/troya/troya.html

ABSTRACT

In this work we present the most important existing developments and future trends in a sub-class of Parallel Genetic Algorithms (PGAs). PGAs are mainly subdivided into coarse and fine grain PGAs, being the coarse grain models (cgpGAs) the most popular ones. An exceptional characteristic of these algorithms is that a cgpGA is not just the parallel version of a sequential algorithm intended to provide speed gains. Instead it represents a new kind of search (optimization) method of higher efficiency and efficacy. The high robustness of these algorithms has led to an increasing number of applications in the fields of artificial intelligence, numeric and combinatorial optimization, business, etc. We make a formalization of these algorithms and present a survey on their most important technical issues. Also useful tables with resumes on their main applications and Internet pointers to important sites are included in order to ease the access of new researchers to this growing area.

Keywords: *Coarse grain parallel genetic algorithms, Evolution schemes, Migration, Classification and characteristics, Overview on technical issues*

A USEFUL REVIEW ON COARSE GRAIN PARALLEL GENETIC ALGORITHMS

ABSTRACT

In this work we present the most important existing developments and future trends in a sub-class of Parallel Genetic Algorithms (PGAs). PGAs are mainly subdivided into coarse and fine grain PGAs, being the coarse grain models (cgpGAs) the most popular ones. An exceptional characteristic of these algorithms is that a cgpGA is not just the parallel version of a sequential algorithm intended to provide speed gains. Instead it represents a new kind of search (optimization) method of higher efficiency and efficacy. The high robustness of these algorithms has led to an increasing number of applications in the fields of artificial intelligence, numeric and combinatorial optimization, business, etc. We make a formalization of these algorithms and present a survey on their most important technical issues. Also useful tables with resumes on their main applications and Internet pointers to important sites are included in order to ease the access of new researchers to this growing area.

1. INTRODUCTION

A genetic algorithm (GA) is a heuristic for searching for an optimum in complex spaces by applying nature-inspired techniques like selection of the fittest, chromosome crossovers or even mutations to a population of candidate solutions (strings of parameters). Every string has an associated fitness value (computed by a function that represents the problem) intended to provide a rank for all the strings [36], [26]. GAs are problem-independent search algorithms (high robustness) with an efficient ratio between visited/total search points that can yield multiple solutions to the problem without additional efforts.

Coarse grain parallel GAs (cgpGAs) [44], [17] need in general a smaller number of steps than sequential GAs to get to the optimum. Hence they are not just a parallel version of a sequential genetic algorithm. In fact cgpGAs actually reach the ideal goal of having a parallel algorithm whose behavior is better than the sum of the separate behaviors of its component sub-algorithms.

Let's begin by a description of the sequential GA first. Since GAs apply operators drawn from nature the nomenclature used in this field is closely related to the terms we can found in biology. Table I resumes these special words.

TABLE I.- NOMENCLATURE

Genotype	The code, devised to represent the parameters of the problem in the form of a string
Chromosome	One encoded string of parameters (binary, gray, floating point numbers, etc...)
Individual	One or more chromosomes with an associated fitness value
Gene	The encoded version of a parameter of the problem being solved
Allele	Basic component of a gene. One or more alleles compose a gene (parameter)
Locus	The position that the allele occupies in the chromosome
Phenotype	Decoded -decimal- version of the genotype, suited for being evaluated
Fitness	Real value indicating the quality of an individual as a solution to the problem
Environment	The problem. It is represented as a function that allows the evaluation of phenotypes
Population	A set of individuals with their associated statistics (fitness average, hamming dist...)
Selection	Policy for selecting one individual from the population (selection of the fittest, ...)
Crossover	Operation that merges the genotypes of two selected parents to yield two new children
Mutation	Operation that changes spontaneously one or more alleles of the genotype

The algorithm operates on a population of strings of problem parameters. Every string is called an *individual* and is composed of one or more *chromosomes* and a *fitness* value. Normally an individual contains just one chromosome (haploid individuals) that represents the set of parameters (variables) called *genes*. Every gene is in turn encoded (usually) in binary by using a given number of *alleles* (0,1). The fitness function behaves as the *environment* to which the decoded version of the *genotype* (called the *phenotype*) is faced in order to know how good it is as a solution.

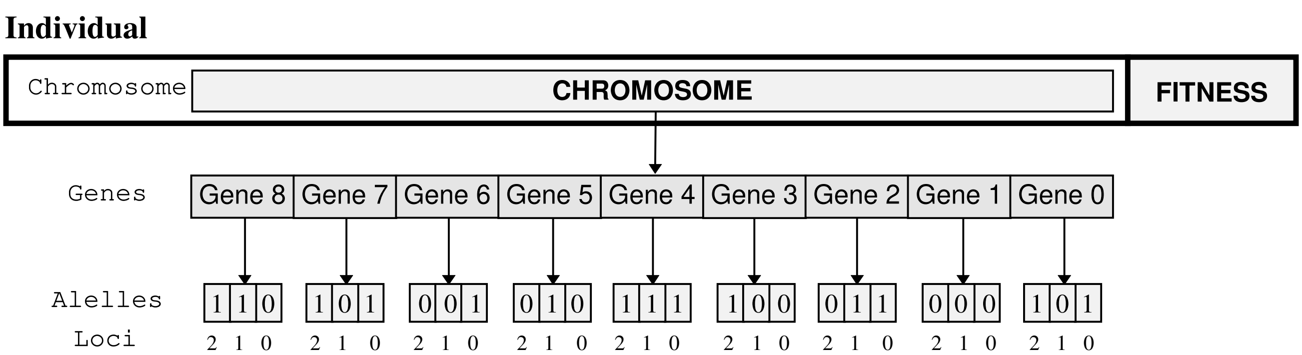

Figure 1. *Some details on the genotype.*

Therefore the population of individuals can be ranked according to the fitness values. Then probabilistic selection of the fittest can be applied to select two parents and cross then up to yield two new children with their mixed and also mutated chromosomes. These operations are repeated until a new population had been built up. The new population needs to be evaluated and the process starts again. The crossover and mutation are applied according to a given probability and not deterministically.

If we mimic natural evolution we won't operate on a single population in which a given individual has the potential to mate with any other one in the entire population (*panmixia*). Instead species tend to reproduce within subgroups. A population distributed among a number of semi-isolated breeding groups is known as *polytypic*. The cgpGA introduces the concept of semi-isolated *demes*. The local selection and reproduction rules locally evolve the species. Diversity is enhanced by migrations.

cgpGAs (and GAs in general) take part of a larger family of evolutionary algorithms (EAs) [11] just like other techniques like Evolutionary Programming (EP) [24] and Evolution Strategies (ES) [58]. The main differences among these algorithms relay on the kind of operations they apply on the strings and the coding of these strings. The term *evolutionary computation* refers to the study of the foundations and applications of these evolutionary heuristics. In spite of the fact that these techniques can be classified into three main categories (we call it "The Natural Equation", see figure 2), this classification is based in minor details and historical development facts rather than in major functioning differences. In fact, their biological basis is essentially the same.

$$\underset{[11]}{\mathbf{EA}} = \underset{[36]}{\mathbf{GA}} + \underset{[24]}{\mathbf{EP}} + \underset{[58]}{\mathbf{ES}}$$

Figure 2. *The Natural Equation.*

Applications of parallel genetic algorithms (PGAs) are growing in number and importance [66], [3], [4], [7]. Also there exist many evidences of the higher efficacy and efficiency of PGAs over traditional sequential GAs [1], [16].

Genetic algorithms are a class of guided random evolutionary algorithms (see figure 3). If we take a deeper look on them we can distinguish among 4 different types of parallelization in PGAs [16]. Types 1 and 2 proceed in the same way than a sequential GA but in a faster manner. This is achieved by either profiting from a code parallelizer embedded in the compiler or else by the explicit parallelization (master-slave *global parallelization*) of the genetic operators and/or evaluations.

Many different models have been proposed for the rest of PGAs [21], [32], [46], [68], [73]. However all these existing models can fit into two more general classes depending of their grain of parallelism called coarse or fine grain parallel GAs (cgpGAs and fgpGAs). This classification relies on the ratio computation/communication. If this ratio is high we call the PGA a coarse grain algorithm and if the ratio is low then we call it a fine grain parallel GA.

Coarse grain PGAs are also known as distributed or island GAs and fine grain PGAs are known as cellular, diffusion or else massively-parallel GAs. The first models for cgpGAs were thought to run on MIMD platforms while fgpGAs were developed to exploit the SIMD parallelism. However this situation has notably changed as we will see in the following sections.

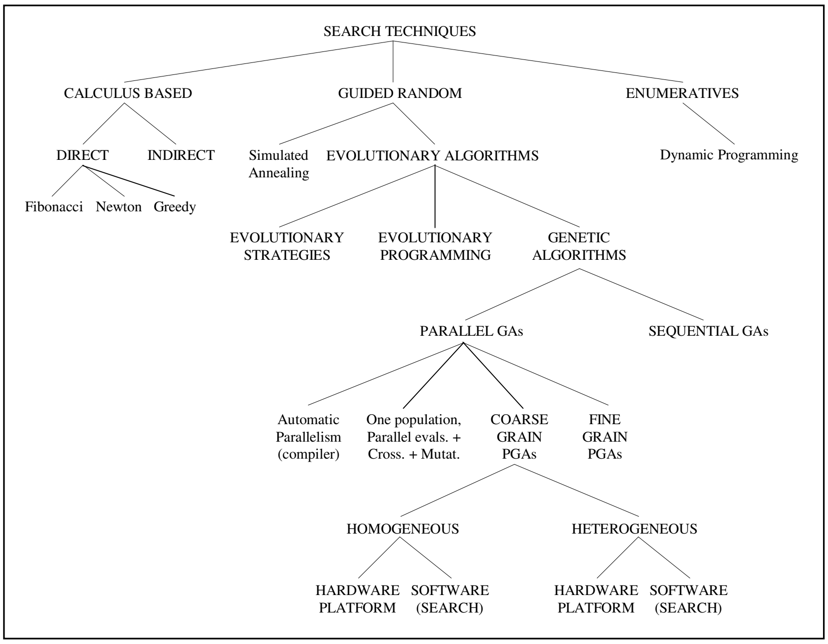

Figure 3.- *A possible tree of search techniques.*

The most representative cgpGA is a ring of sub-populations running in parallel with sparse migrations (copies) of individuals among them. On the other hand the most popular fgpGA is a toroidal grid of individuals each one interacting only with its neighborhood.

A PGA can run on a network of computers or else in massively parallel computers, with independence of its granularity. This provokes a further classification depending on the homogeneity of the hardware. On the other hand the sub-populations belonging to a cgpGA (or the local strings in a fgpGA) can evolve by applying the same techniques or else by using different kinds of coding, operators, parameters, etc... This last motivates our distinction on the homogeneity at the software level.

We can resume the advantages of using cgpGAs in the following table.

TABLE II.- ADVANTAGES OF USING A cgpGA

Advantages of using a Coarse Grain Parallel Genetic Algorithm	
❶ Work on a coding of the problem (least restrictive)	❺ Easy parallelization as islands of sequential GAs
❷ Basically independent of the problem (robustness)	❻ Loosely coupled (low communication overhead)
❸ Can yield alternative solutions to the problem	❼ Higher efficiency and efficacy than sequential GAs
❹ Parallel search from multiple points in the space	❽ Easy cooperation with other search procedures

2. HOW cgpGAs WORK

In this section we want to present two complementary views of a cgpGA. First we address a global characterization by outlining the pseudocode of a canonical cgpGA and then we show in the second subsection a formal description of the canonical cgpGA.

2.1. General Description of a cgpGA

A cgpGA proceeds in an iterative manner by generating new populations from old ones. It applies selection, crossover and mutation on several initially random populations in order to compute the new ones, and then proceeds iteratively on them. These operators are not applied deterministically but they have an associated probability of being applied (usually high for crossover and small for mutation) [64]. In figure 4 you can see the basic pseudocode of a traditional cgpGA.

In the canonical model, the sub-populations exchange individuals with a given frequency called the *migration gap*. Alternatively a probability of making migrations can be used in every sub-population (on the average, with a long evolution, it shouldn't appear differences). The sub-algorithms exchange else a copy of individuals [14] or the individuals themselves [45].

An open research question is how often migrations should take place and also how many individuals should encompass this jump between populations. The number and location of the target sub-populations define the topology of the cgpGA and its global behavior.

Frequencies of migrations are usually small, just as the number of individuals that undertake migration. Relating the topology, trees, stars, hypercubes and may more have been studied (although the most usual one is a unidirectional ring of processes) [15]. Fixed topologies are much more usual than dynamic ones. Finally we need clear policies for selecting which individuals are going to migrate and which ones are going to be replaced by the incoming migrants. Usually the individuals to migrate are selected randomly from the present population and the algorithm only replaces one individual of its population in the immigrant has a higher fitness than it (see section 5.6).

```
GA(i):
P(t=0) := (i1, ..., in);    // Generate the initial pop.
WHILE not termination_condition DO
   Generate  P1(t)  from  P(t) by application of the
        selection operator on P(t);
   Generate P2(t) from P1(t) by application of the
        crossover operator with probability Pc on P1(t);
   Generate P3(t) from P2(t) by application of the
        mutation operator with probability Pm on P2(t);
   Send migrants from P3(t) to neighbor sub-populations;
   Get migrants into P3(t) from neighbor sub-populations
        to yield P4(t);
   P(t+1) := P4(t);
   t := t + 1;
END WHILE;
solution = Best_individual_Ever_Found;
```

Figure 4. *Pseudocode of a simple coarse grain parallel GA.*

If we analyze the basic operations we will notice that the selection operator is usually like a roulette wheel spin intended to get good parents. The crossover consists in selecting one point in the two strings and exchanging the resulting slices. The mutation is a bit flip operation. A big number of variants exist for these and other operators. The most important of them are revised in section 5.

String#	String	Fitness	% of the Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		1170	100.0

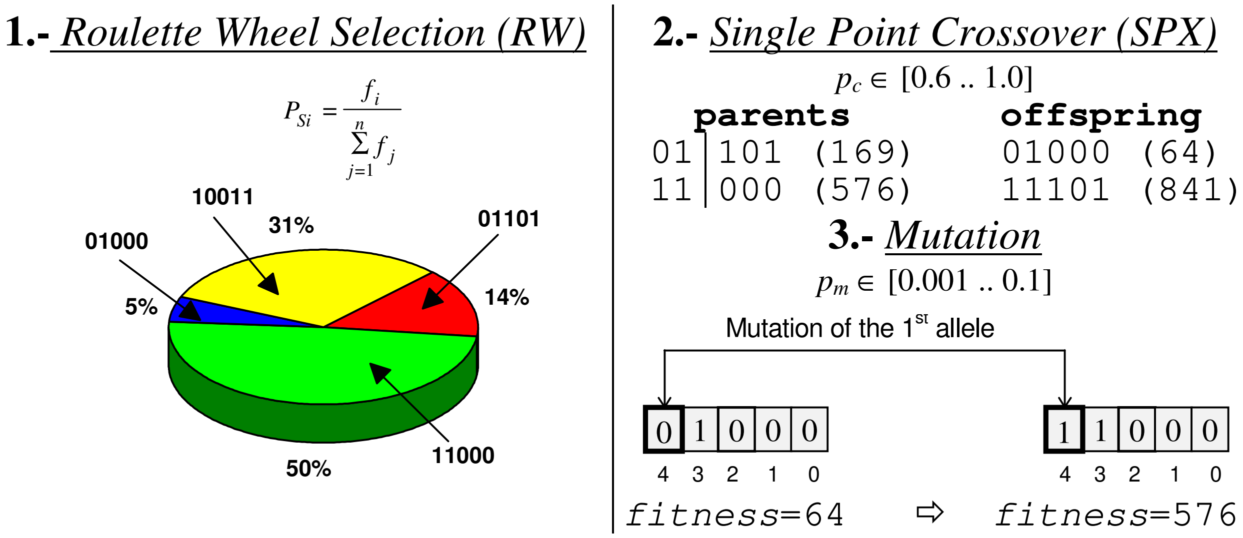

Figure 5.- *An example of the computations that a cgpGA performs in every sub-algorithm to maximize $f(x)=x^2$. The parameter x is encoded as a string of five bits in binary. The population size is four strings.*

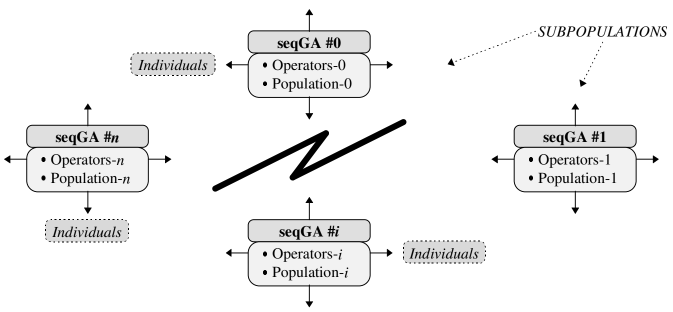

Figure 6.- *A typical description of a cgpGA.*

The canonical cgpGA can be thought of consisting in several sub-algorithms running in parallel with sparse (synchronous or asynchronous) exchanges of individuals. They work on a given topology

2.2. Formal Description of a Canonical cgpGA

In this section we are going to show the formal aspect of a canonical cgpGA. This mathematical description is intended to help to capture the common points of any cgpGA but also to highlight the points in which extensions or hybridizations can be derived from.

ALGORITHM GA$_i$, $\forall i \in \{0,\ldots,d\text{-}1\}$

$t^i := 0$; *//Super-index ^{i} always means i-th population*

initialize: $P^i(0) := \{\vec{a}_1^{\,i}(0),\ldots,\vec{a}_{Ni}^{\,i}(0)\} \in G^{Ni}$ **where** $G^{Ni} = \{0,1\}^l$; *//Generate a random initial population*

evaluate: $P^i(0) : \{\Phi(\vec{a}_1^{\,i}(0)),\ldots,\Phi(\vec{a}_{Ni}^{\,i}(0))\}$ **where** $\Phi(\vec{a}_k^{\,i}(0)) := \varphi\left(f\left(\Gamma\left(\vec{a}_k^{\,i}(0)\right)\right), P^i(0)\right)$; *//Compute fitness*

while $\left(T_i\left(P^0(t),\ldots,P^{d-1}(t)\right) \neq \mathbf{true}\right)$ **do** *//While not termination condition do*

- *select* : $P^{i'}(t) = s\left(P^i(t)\right)$ **where** $p_s\left(\vec{a}_k^{\,i}(t)\right) := \Phi\left(\vec{a}_k^{\,i}(t)\right) \Big/ \sum_{j=1}^{Ni} \Phi\left(\vec{a}_j^{\,i}(t)\right)$;
- *recombine* : $\vec{a}_k^{\,i'}(t) := c'_{\{pc\}}\left(P^{i'}(t)\right) \quad \forall k \in \{1,\ldots,Ni\}$;
- *mutate* : $\vec{a}_k^{\,i''}(t) := m_{\{p_m\}}\left(\vec{a}_k^{\,i'}(t)\right) \quad \forall k \in \{1,\ldots,Ni\}$;
- *evaluate* : $P^{i''}(t) := \left\{\vec{a}_1^{\,i''}(t),\ldots,\vec{a}_{Ni}^{\,i''}(t)\right\} : \left\{\Phi\left(\vec{a}_1^{\,i''}(t)\right),\ldots,\Phi\left(\vec{a}_{Ni}^{\,i''}(t)\right)\right\}$ *//Compute fitness*
  where $\Phi\left(\vec{a}_k^{\,i''}(t)\right) = \varphi\left(f\left(\Gamma\left(\vec{a}_k^{\,i''}(t)\right)\right), P^i(t-\omega)\right)$;
- *replace* : $P^{i'''}(t) := r_{\{pr\}}\left(P^i(t), P^{i''}(t)\right)$;
- *set_migrant* : $\vec{a}_m^{\,i}(t) := M^i\left(P^{i'''}(t)\right) \quad \forall m \in \{1,\ldots,mr\}$; *//Select* `mr` *migrants*
- *migrate* : $P^i(t+1) := r_{\{p_M\}}\left(P^{i'''}(t), \vec{a}_m^{\,z}(t)\right)$ **where** $z := \upsilon(i,d)$;
- $t^i := t^i + 1$;

end_while;

MEANING OF SYMBOLS

❖ $f : \Re^{gn} \to \Re$ $\quad f : \prod_{j=1}^{gn}\left[u_{ji}, v_j\right] \to \Re\left(u_j < v_j\right)$ $\quad \Phi : G \to \Re$ $\quad$ *gn*: genes' number

The fitness function *f* assigns a real value to every string. Since we suppose a maximization problem we use the **Φ** function to ensure positive values and that better solutions will have higher fitness values.

❖ $\Gamma_j\left(a_{j1},\ldots,a_{jlx}\right) := u_j + \dfrac{v_j - u_j}{2^{lx}-1} \cdot \left(\sum_{r=1}^{lx} a_{jr} \cdot 2^{r-1}\right)$ normally $u_j = u \,|\, \forall j \in \{1,\ldots,gn\},\ v_j = v \,|\, \forall j \in \{1,\ldots,gn\}$

Decoding a string means computing the decimal value of the binary representation of every gene $\vec{a}_j$. Usually all the parameters in string use the same number *lx* of bits and decode to the same real range.

❖ $\varphi\Big(f\Big(\Gamma\Big(\vec{a}_k^{\,i}(t)\Big)\Big)\Big)\geq 0 \qquad \varphi\Big(f\big(\Gamma(\vec{a})\big), P^i(t-\omega)\Big) = max\Big\{ f\Big(\Gamma\Big(\vec{a}_j\Big)\Big) \;\Big|\; \vec{a}_j \in P^i(t-\omega)\Big\} - f\big(\Gamma(\vec{a})\big)$

After decoding every parameter in the string and computing the fitness we scale with φ in order to ensure positive growing values by using the maximum fitness found in the last ω algorithmic steps.

❖ υ**: IxI → I** **RING**: $\upsilon(i,d) := (i+1)\,\mathbf{mod}\,d$ **TREE**:... **HYPERCUBE**:...

This is the topology function that allows to compute the neighbor population. It can be extended to return a set of neighbor populations instead of only one.

❖ $m_{\{p_m\}}: \mathbf{G} \rightarrow \mathbf{G} \quad m_{\{p_m\}}\big(s_1,\ldots,s_l\big) = \big(s_1',\ldots,s_l'\big) \;\; \forall j \in \{1,\ldots,l\} \;\; s_j' = \begin{cases} s_j & \alpha_i > p_m \\ 1-s_j & \alpha_i \leq p_m \end{cases} \quad \alpha_j \in [0,1]$

The mutation inverts every bit of the string relaying on a uniform random variable α_j sampled anew for every string position.

❖ $c_{\{p_c\}}: \mathbf{G}\mathrm{x}\mathbf{G} \rightarrow \mathbf{G}\mathrm{x}\mathbf{G} \qquad \tau \in [0,1] \qquad \alpha \in \{1,\ldots,l\}$

$$\textbf{if } \tau > p_c \textbf{ then } c_{\{p_c\}}\big(\vec{a}=(a_1,\ldots,a_l),\vec{b}=(b_1,\ldots,b_l)\big) = \big((a_1,\ldots,a_\alpha,b_{\alpha+1},\ldots,b_l),(b_1,\ldots,b_\alpha,a_{\alpha+1},\ldots,a_l)\big)$$

$$\textbf{else } c_{\{p_c\}}\big(\vec{a}=(a_1,\ldots,a_l),\vec{b}=(b_1,\ldots,b_l)\big) = \big((a_1,\ldots,a_l)(b_1,\ldots,b_l)\big)$$

The crossover operator admits many variants. The basic one is called single point crossover (SPX). It exchanges two portions of every parent to yield two new offspring. In the algorithm we use the notation (c') to express that the crossover should be extended to the whole population.

❖ $r_{\{p_M\}}\Big(P^i(t),\vec{a}_m^{\,z}(t)\Big) := \begin{cases} r\Big(P^i(t),\vec{a}_m^{\,z}(t)\Big) & \alpha > p_M = \frac{1}{mig\_gap} \\ P^i(t) & \alpha \leq p_M = \frac{1}{mig\_gap} \end{cases} \qquad \alpha \in [0,1]$

This is an example of the replacement operator that uses a given *r()* policy to choose an individual from the population. It takes place (i.e., migration takes place) or not depending on the random variable α.

Some parameters as the size of every sub-population (N_i) or the migration policy (M^i) are left indicated to depend on the considered sub-algorithm. Besides the probabilities of mutation p_m and crossover p_c other important parameters are: the number of migrants (*mr*), the frequency of migrations (*mig_gap*) and the topology ($\upsilon(i,d)$). Some variants can be derived by having different values for all these parameters in every component sub-algorithm (see section 5.7 for more details).

3. THEORETICAL EXPLANATION OF WHY cgpGAs WORK

In this section we present the most important results that formalize the work of a cgpGA. First we analyze the schema theorem [49] (see also *the response to selection* theory [50]) as the more popular explanation of the kind of search that a single GA performs.

Then we will address the extensions for a cgpGA algorithm. Finally we review a figure of merit to characterize the search in terms of the number of schemata and the convergence time.

3.1. The Schema Theorem

The basic work of any GA resides in the implicit management of *schemata*. Any individual (string) is considered to be an instance of several schemata. If we are working with strings on a binary alphabet **V**={*0*,*1*}, a schema is a hyperplane of the search space that can be represented as a string of the extended alphabet **V+**={*0*,*1*,*}. The '*' symbol indicates a *don't care* (undefined) position in the schema.

Any given string in the GA is an instance of a large number of schemata. For example the strings *s1*=1110 and *s2*=1100 are possible instances of the schema *S*=11*0. Good schemata have a small number of defined (non '*') positions *o(S)* -*order*- and the distance $\delta(S)$ between the first and last defined positions -*defining length*- is short. This kind of schemata have a lower probability of being corrupted and therefore to be mixed with another good schemata in order to reach a solution.

- *S1*=011*1**, *S2*=0******, *o(S1)*=4 y *o(S2)*=1, $\delta(S1)$=5-1=4, $\delta(S2)$=1-1=0.

Holland [36] initial stimates work out a result known as the implicit parallelism of a GA. This effect consists in that although the GA is working on *n* strings, it is really working with $O(n^3)$ schemata in the schema space (**implicit parallelism** property of GAs). Every binary string is an instance of up to 2^l schemata. A population of *n* individuals contains a maximum of $n \cdot (2^l)$ schemata.

Although this basic description still holds for binary alphabets, there exists a new interpretation of schemata that overturns the drawbacks of low expressiveness that this initial interpretation assigns to alphabets of higher cardinality (for example **V=I** or even **V=**$\Re$) [9].

In the following figure we resume the additive effects that every operation (selection, crossover, mutation) provokes on the number of instances (strings) belonging to a given schema.

• <u>THE EFFECTS OF SELECTION</u> **(A)**

1.- $m(S,t+1) = m(S,t) \cdot \dfrac{f(S)}{\bar{f}}$ (If *f(S)* over average then...)

2.- $m(S,t+1) = m(S,t) \cdot \dfrac{\bar{f} + c \cdot \bar{f}}{\bar{f}} = (1+c) \cdot m(S,t)$

3.- $m(S,t) = m(S,0) \cdot (1+c)^{t}$ (The fittest schemata grow up)

• <u>THE EFFECTS OF CROSSOVER</u> **(B)**

1.- $P_s(S) \geq 1 - P_c \cdot \dfrac{\delta(S)}{l-1}$ (Survival probability)

2.- $m(S,t+1) \geq m(S,t) \cdot \dfrac{f(S)}{\bar{f}} \cdot \left[1 - \left(P_c \cdot \dfrac{\delta(S)}{l-1} \right) \right]$

• <u>THE EFFECTS OF MUTATION</u> **(C)**

1.- $P_s(S) = \left(1 - P_m\right)^{o(S)}$

2.- $P_m << 1 \Rightarrow \left(1 - P_m\right)^{o(S)} \approx 1 - o(S) \cdot P_m$ (Survival probability)

• <u>THE SCHEMA THEOREM</u> **(D)**

$$m(S,t+1) \geq m(S,t) \cdot \frac{f(S)}{\bar{f}} \cdot \left\{ 1 - \left(P_c \cdot \frac{\delta(S)}{l-1} \right) - o(S) \cdot P_m \right\}$$

Figure 7.- *The schema theorem predicts the number of instances m(S,t+1) of a schema S at time t+1.*

This theorem explains why a good solution schema will receive an exponentially increasing number of strings (*m(S,t)*) as the evolution proceeds. *Pc* and *Pm* are the probabilities of applying crossover and mutation, *f(S)* is the schema average fitness and $\bar{f}$ is the population average fitness.

Under proportionate selection (fig. 7.A) every above-average fitness schema gets an exponential number of instances throughout generations (*t*). Under crossover (fig. 7.B) only short schemata are appropriately sampled in the next generation (depending of the defining length of the schema and the probability of applying crossover) -also construction of new and existing schemata takes place-. Finally only low order schemata have a high probability of surviving to the effect of mutation.

Hence the selection defines the fittest strings, crossover samples new points in the problem space and finally the mutation gets new alleles into the population (and corrupt some of the existing ones) in order to get out of local optima and allow smooth changes for final tuning.

Hence if a problem can be expressed in terms of basic building blocks on some kind of alphabet (floating point or integer numbers are another popular kind of alphabets in GAs and EAs) and a fitness function for strings can be defined then we are very likely able to solve this problem with GAs.

3.2. Extensions of the Schema Theorem for the cgpGA

We can consider cgpGAs as efficient and robust as their sequential counterparts if they are showed to allocate better and better strings to good schemata by following also an exponential law.

If P_m is the probability of mutation on every allele (string position) in all the distributed nodes of the cgpGA then a schema S of order $o(S)$ survives in the whole population with probability $1\text{-}P_m o(S)$, just like in the sequential GA. Also it can be show that the probability of being successfully recombined and the selection growing rate for good schemata are basically the same than for the sequential algorithm. We only need to supposed that the average fitness in all the sub-algorithms is approximately the same. See [29] for a complete development of this result.

We must note that this last guess is only correct for synchronous cgpGAs or else asynchronous ones running in computers of similar characteristics. For a softw/hardw heterogeneous cgpGA this assumption might not hold.

Of course these results only mean that the cgpGA is an efficient and robust search procedure. The actual performance of this algorithm is usually considerably larger than the sequential GA. Not only we have sub-algorithms with smaller populations that run every step in parallel (reducing the whole processing time) but also this kind of search maintains samples of very different good zones of the search space in every population thus showing a higher efficacy and probability of getting a solution.

3.3 Optimum Schema Processing Rates in a cgpGA

There exists an interesting measure of the relative merit of any parallel GA developed by Goldberg [27] that relates the schema processing rates to the time of convergence of the algorithm and the parallel disposition of the individuals. The results presented in this section are actually enhanced by the fact that a cgpGA is loosely coupled (sparse migrations) and does not degrade the performance of the algorithm.

If we had a perfectly parallel machine with one processor for every individual then the convergence time of the algorithm would be constant $t_c \in \Theta(1)$ relating the length of strings. But the communication delays and other time consuming operations will slow down this ideal value.

If φ is the ratio between the parallel and sequential speeds and if we suppose it can be expressed as a power of the number of processors $\varphi = n^{\beta}$ then β is the parallelization degree. Hence the time of convergence can be expressed as $t_c = n^{1-\beta}$ where $\beta=0$ is the sequential GA and $\beta=1$ represents the perfectly parallel algorithm. Therefore we can define a figure of merit by relating the number of schemata processed in the whole evolution (n_c is the number of steps and t_c is the time for one step):

$$M(n,l,\beta) = \frac{\Delta S}{\Delta t} = \frac{S(n,l) - 2^l}{(K \cdot n \cdot \mathbf{log}\, n) \cdot n^{1-\beta}} \quad \textbf{(eq. 1)} \qquad M(n,l,\beta) = \frac{2^l \cdot (n-1)}{(K \cdot n \cdot \mathbf{log}\, n) \cdot n^{1-\beta}} \quad \textbf{(eq. 2)}$$

where K is a constant for the actual number of generations and $n \cdot \mathbf{log}\, n$ is the number of generations of the algorithm to converge to a solution (worst case). For medium size populations, the *schema function* $S(n,l)$ can be estimated to $S(n,l) \approx n \cdot 2^l$ and the right hand equation gives us the merit of a parallel GA depending on the size of the population (n), the length of the strings (l) and the degree of parallelism (β).

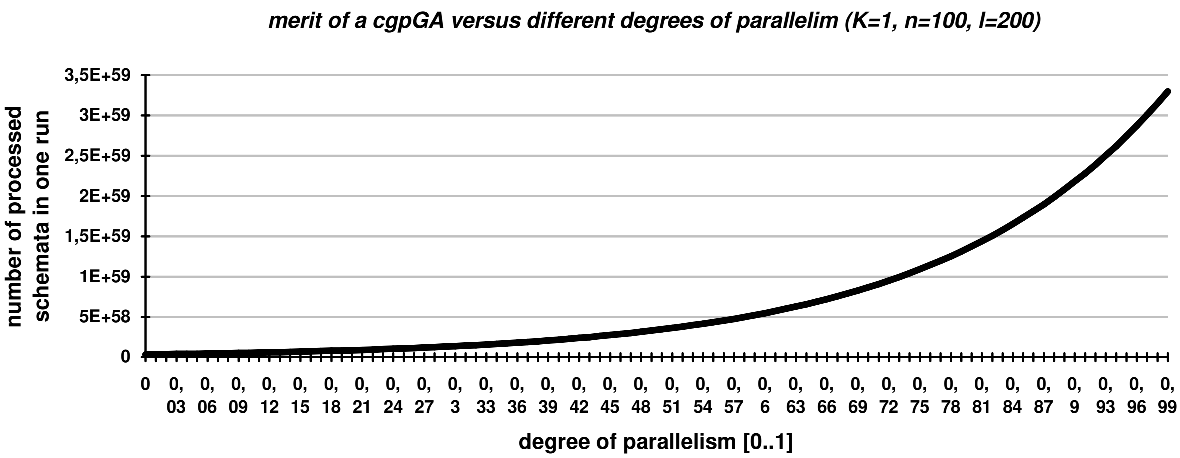

Figure 8.- *Example application with 100 strings of length 200 bits. Relationship between the merit and the degree of parallelism.*

The relative merit of a sequential GA (β=0) versus a perfectly parallel GA (β=*1*) is *M(n,l,0)/M(n,l,1)≈1/n*. This means that the parallel version is *n* times better than the sequential one. On the other hand the degree of parallelism needed to get twice the merit of a sequential GA is:

$$\beta_{twice} = \frac{\log 2 - \log n}{\log n} + 1 \qquad \textbf{(eq. 3)}$$

This means that this degree of parallelism depends on the number of strings in the population and that values as small as β=0.2 can easily allow to duplicate the performance of a sequential algorithm. We do not need ideally perfect parallel algorithms to get a high performance in a cgpGA.

4. CLASSIFICATION OF PARALLEL AND SEQUENTIAL GAs

In this section we make a survey of the algorithms and software more readily accessible in this area. Let's begin with a timely review of cgpGAs. The most known cgpGAs sorted by approximate date of appearance are:

TABLE III.- TIMELY OVERVIEW OF COARSE GRAIN PARALLEL GAs

cgpGA	Ref.	Year	Main Characteristics
PGA	[55]	1987	Generational cgpGA on a Intel iPSC hypercube of 8 CPUs. Migrate the best. Dynamic Topol.
dGA	[68]	1989	Distributed populations. Good results with 20% of population migrants every 20 generations
GENITOR II	[73]	1990	Steady-State islands with ranked selection and reduced surrogate crossover
PGA	[32]	1991	Sub-populations in a circular ladder-like 2-D topology. Migrate the best, local hill-climbing
SGA-cube	[23]	1991	Made for nCUBE2. It is the parallel extension of the well known simple GA of Goldberg
PARAGENESIS	---	1992	Made for the CM-200. It places one individual in every CPU
PeGAsuS	[59]	1993	Targeted for MIMD machines and written in a very high and flexible description language
GAMAS	[56]	1994	Uses 4 very heterogeneous species (islands) and quite specialized migrations and genotypes
iiGA	[44]	1994	Injection island GA with hierarchical heterogeneous nodes and asynchronous migrations
SP1-GA	[42]	1994	128 steady-state islands on a IBM SP1 machine of 128 nodes. 2-D toroidal mesh. mr=1
DGENESIS	[47]	1994	Free topology, mr and m_gap and policies for selection. Implemented with sockets and UDP
GALOPPS	[30]	1996	Very flexible. Implemented with PVM and comprising a large number of operators
GDGA	[45]	1996	Synchronous. Simulated on one processor. Generational. Uses Fuzzy crossover and FP genes
CoPDEB	[2]	1996	Every islands uses its own probabilities for mutation and crossover and specialized operators

This list is not complete of course, but it gives us a filling of the evolution of cgpGAs through the recent years. We will offer more details on those and other algorithms in the following tables.

There's a tendency during this last years towards the making of cgpGAs that run on clusters of machines and that pay a great attention to their portability. Old systems like transputer machines do not support recent implementations. Also new developments in JAVA will appear as a confirmation of these tendencies.

Next we present a set of representative applications of cgpGAs in order to show the wide area of successful efficiency (i. e. robustness) of these algorithms.

TABLE IV.- SOME APPLICATIONS OF COARSE GRAIN PARALLEL GAs

Reference	**Application Domain**
[7]	Parallel training of artificial neural networks. Hybridized and pure cgpGAs
[19]	Synthesis of VLSI circuits
[31]	Function optimization
[42]	Set partitioning problem
[44]	Graph partitioning problem
[49]	Constraint Optimization, reordering problems, ...
[51]	Traveling salesperson problem (TSP), function optimization
[53]	Distributing the computing load onto a set of processing nodes
[56]	The file allocation problem, XOR neural network, sine envelope sine wave function
[66]	Systems modelling, protein tertiary structure prediction and two-dimensional bin packing problems
[68]	Walsh polynomials
[73]	Optimization of the connection weights of neural networks (XOR, bin-adder, ...) and function optimization

There exist many other applications in very interesting domains as frequency assignment problems, market predictions, game theory, philters, graphic transformations, etc... Some of these applications motivated the needed of new classes of genotypes that can be thought of consisting in strings of arbitrary symbols. A further extension led to the so called *genetic programming* paradigm [40] in which the individuals are *trees of symbols* representing computer programs, fuzzy controllers or many other kind of high-level concepts. It is a new an important research area in which cgpGAs are developing fast.

Returning back to the spirit of this section we can classify any existing algorithm into three major categories [59] according to their specific objectives:

• **Application Oriented**: These are black-box systems designed to hide the details of GAs and to help the user in developing applications for specific domains. Some of them are useful for very different purposes as scheduling or telecommunications (e.g. PC/BEAGLE). Some others are much more application oriented (like OMEGA for finance). Usually they are menu-driven an easily parameterizable.

• **Algorithm Oriented**: Based on specific algorithms. The source code is available in order to provide an easy incorporation into new applications. This class may be further sub-divided into:

- **Algorithm Specific**: They contain one single GA (e.g. GENESIS). Their users are else system developers (for making applications) or else researchers in this field (interested in extensions).

- **Algorithm Libraries**: They support a group of algorithms in a library format (e.g. OOGA). They are highly parameterized and contains many different operators to ease future applications.

• **Tool Kits**: These are flexible environments for programming a range of GAs and applications. They can be sub-divided into:

- **Educational**: Used for introducing GA concepts to novice users (e.g. GA Workbench). They have easy management of the basic techniques to track executions and results during the evolution.

- **General Purpose**: Useful for modifying, developing and supervising a wide range of operators, algorithms and applications (e.g. Splicer).

We show some detailed tables with a classification of algorithms. We note this classification with the above labels. Also we use the PROP label for identifying commercial and proprietary systems. We have open the contents of this table to also non-cgpGAs to offer a better landscape of the global achievements of this class of evolutionary algorithms.

TABLE V.- OVERVIEW ON SOME UP-TO-DATE EXISTING SEQUENTIAL AND PARALLEL GENETIC ALGORITHMS

Genetic Algorithm	Ref.	Lang.	OS/Machine	Kind of GA	Algorithm	Seq/Par	Evolution	HOW TO GET IT!
ASPARAGOS	[32]	C	64 T800 transp	Algorithm Specific	Single / No pop.	PAR	Local	---
CoPDEB	[2]	C	UNIX	Algorithm Specific	Single / No prop.	PAR	Gen.	mail to: adamidis@it.teithe.gr
DGENESIS 1.0	[47]	C	UNIX	Algorithm Specific	Single / No Prop.	PAR	Percentage	ftp to: ftp.aic.nrl.navy.mil
ECO-GA	[21]	C?	UNIX?	Algorithm Specific	Single / No Prop.	PAR	Local	mail to: yuval@wisdom.weizmann.ac.il
EM 2.1	[69]	C	DOS	Algorithm Library	Algrthm. Library	SEQ	Gen. / S. S.	ftp to: [130.149.192.50]/⌖
EnGENEer	[60]	C	UNIX	General Purpose	Algrthm. Library	SEQ/par	Gen. / S. S.	---
ESC^{A}P$_{A}$DE 1.2	[35]	---	---	Algorithm Specific	Single / No Prop.	---	E. S.	ftp to: ls11.informatik.uni-dortmund.de
EVOLVER 2.0	---	C++	DOS & MAC	Application Orient.	Single / Prop.	SEQ	---	Axcélis (used in a spread sheet)
GA Workbench	[37]	C	DOS	Educational	Algrthm. Library	SEQ	Percentage	ftp to: camcon.co.uk
GAGA	[8]	C	UNIX	Algorithm Specific	Single / No Prop.	SEQ	---	ftp to: cs.ucl.ac.uk/darpa/gaga.shar
GAGS	[48]	C++	UNIX & DOS	General Purpose	Algrthm. Library	SEQ	Gen.	http://kal-el.ugr.es/GAGS
GAlib	[71]	C++	UNIX	Algorithm Specific	Algrthm. Library	SEQ	Gen.	ftp to: lancet.mit.edu/pub/ga/galib242.tar.gz
GALOPPS 3.1	[30]	C	UNIX & DOS	Algorithm Specific	Single / No Prop.	PAR	Gen.	ftp to: isl.msu.edu/pub/GA
GAMAS	[56]	C?	UNIX	Algorithm Specific	Single / No Prop	PAR	Gen.	---
GAME	[66]	C++	UNIX & DOS	General Purpose	Algrthm. Library	SEQ/par	Gen. / S. S.	ftp to: bells.cs.ucl.ac.uk/papagena/game
GAucsd 1.2 / 1.4	[61]	C	UNIX, DOS ...	Algorithm Specific	Single / No Prop.	SEQ	Gen.	http://www.aic.nrl.navy.mil/galist/src/GAucsd14.sh.Z
GDGA	[45]	C	UNIX	Algorithm Specific	Single / No Prop.	SEQ/par	Gen.	mail to: lozano@decsai.ugr.es
GENESIS 5.0	[33]	C	UNIX, DOS ...	Algorithm Specific	Single / No Prop.	SEQ	Percentage	http://www.aic.nrl.navy.mil/galist/src/genesis.tar.Z
GENEsYs 1.0	---	C	UNIX	Algorithm Specific	Single / No Prop.	SEQ	Percentage	[129.217.36.140]/pub/GA/src/GENEsYs-1.0.tar.Z
GENITOR I - II	[73]	C	UNIX	Algorithm Specific	Single / No Prop.	SEQ-PAR	S. S. / Rank	*/genitor
HSDGA	[70]	C	HELIOS	Algorithm Specific	Single / No Prop.	PAR	Local	---
LibGA	[18]	C	UNIX	Algorithm Specific	Single / No Prop.	SEQ	Gen. / S. S.	*/libga
OMEGA	[66]	---	---	Application Orient.	Prop.	SEQ	---	KiQ Limited (used in business applications)
OOGA	[22]	CLOS	any LISP	Algorithm Library	No Prop.	SEQ	Gen. / S. S.	TSP, P.O. Box 991, Melrose, MA 02176-USA
PARAGENESIS	---	C*	CM-200	Algorithm Specific	Single / No Prop	PAR	Percentage	[192.26.18.74]/pub/galist/src/ga/paragenesis.tar.Z
PC/BEAGLE	[59]	---	DOS	Application Orient.	Prop.	SEQ	S. S.	Pathway Research Ltd. (used in machine learning)
PeGAsuS	[59]	C	UNIX, ...	General Purpose	Algrthm. Library	PAR	Gen. / S. S.	---
PGA 2.5	---	C	UNIX & DOS	Algorithm Specific	Single / No Prop	PAR	Gen.?	http://www.aic.nrl.navy.mil/galist/src /pga-2.5.tar.Z
PGAPack	[43]	C	UNIX, ...	Algorithm Library	Algrthm. Library	PAR	Gen. / S. S.	http://www.mcs.anl.gov/pgapack.html
RPL2	[57]	C	UNIX, ...	General Purpose	Prop. Library	PAR	Gen. / S. S.	http://www.quadstone.co.uk/~rpl2
SGA	[26]	C	UNIX	Algorithm Specific	Single / No Prop	SEQ	Gen.	ftp to: [192.26.18.56] ... sgac_94m.tgz
SGA-Cube	[23]	C	nCUBE 2	Algorithm Specific	Single / No Prop	PAR	Gen.	ftp to: [192.26.18.56] ... sgacub94.tgz
SPLICER 1.0	[54]	C	UNIX & MAC	General Purpose	Algrthm. Library	SEQ	Gen.	ftp to: galileo.jsc.nasa.gov
SUGAL 2.0	[38]	C	UNIX, DOS, ...	Algorithm Library	Algrthm. Library	SEQ	Gen. / S. S.	http://osiris.sund.ac.uk/ahu/sugal/home.html
TOLKIEN	---	C++	UNIX	General Purpose	Algrthm. Library	SEQ	Gen.?	*/tolkien
XpertRule GenAsys	---	---	---	Application Orient.	Prop.	SEQ	---	Attar Software (used in design and scheduling)

TABLE VI.- MEANING OF THE SYMBOLS

Symbol	Meaning of the Symbol
Local	The algorithm performs operations on every string by interactions with only its neighboring strings
Gen.	Generational: The basic algorithmic step is a full generation of individuals
S. S.	Steady-State: the basic algorithmic step is the computation of a very low number of individuals (usually one)
Percentage	GAP: The algorithm works and replaces only a given percentage of the whole population of individuals
Rank	The individuals are sorted according to increasing fitness and the selection uses the 'position' in the rank (and not the fitness itself)
par	Parallel version is not available yet
SEQ/PAR	The algorithm is able to work either in sequential and in parallel
*	`http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/systems`
ϕ	`pub/software/Evolution_Machine/em_tc.exe`

TABLE VII.- CLASSIFICATION OF THE GENETIC SYSTEMS

<table>
<tr><th rowspan="2">Application Oriented</th><th colspan="2">Algorithm Oriented</th><th colspan="2">Tool Kits</th></tr>
<tr><th>Algorithm Specific</th><th>Algorithm Libraries</th><th>Educational</th><th>General Purpose</th></tr>
<tr><td rowspan="5">EVOLVER 2.0</td><td>ASPARAGOS</td><td rowspan="4">EM 2.1</td><td rowspan="19">GA Workbench</td><td rowspan="2"></td></tr>
<tr><td>CoPDEB</td></tr>
<tr><td>DGENESIS 1.0</td><td rowspan="2">EnGENEer</td></tr>
<tr><td>ECO-GA</td></tr>
<tr><td>ESC^{A}P$_A$DE 1.2</td><td rowspan="4">OOGA</td><td rowspan="2">GAGS</td></tr>
<tr><td rowspan="5">OMEGA</td><td>GAGA</td></tr>
<tr><td>GAlib</td><td rowspan="2">GAME</td></tr>
<tr><td>GALOPPS 3.1</td></tr>
<tr><td>GAMAS</td><td rowspan="5">PGAPack</td><td rowspan="2">PeGAsuS</td></tr>
<tr><td>GAucsd 1.2 / 1.4</td></tr>
<tr><td rowspan="5">PC/BEAGLE</td><td>GDGA</td><td rowspan="2">RPL2</td></tr>
<tr><td>GENESIS 5.0 - GENEsYs 1.0</td></tr>
<tr><td>GENITOR I Y II</td><td rowspan="2">Splicer 1.0</td></tr>
<tr><td>HSDGA</td><td rowspan="6">SUGAL 2.0</td></tr>
<tr><td>libGA</td><td rowspan="2">TOLKIEN</td></tr>
<tr><td rowspan="4">XpertRule GenAsys</td><td>PGA 2.5</td></tr>
<tr><td>PARAGENESIS</td><td rowspan="3"></td></tr>
<tr><td>SGA</td></tr>
<tr><td>SGA-Cube</td></tr>
</table>

TABLE VIII.- DETAILS ON THE PARALLEL GAs

Parallel GA	Kind of Parallelism	Topology	Present Applications
ASPARAGOS	Fine grain. Applies Hill-Climbing if no improvement	Ladder	TSP
CoPDEB	Coarse grain. Every sub-pop. applies different operators	Full Connected	Func. Opt. and ANN's
DGENESIS 1.0	Coarse grain with migrations among sub-populations	Any Desired	Function Optimization
ECO-GA	Fine grain. One of the first of its class	Grid	Function Optimization
EnGENEer	Global parallelization (parallel evaluations)	Master / Slave	Very Different
GALOPPS 3.1	Coarse grain. A very portable software	Any Desired	Func. Opt. and Transport
GAMAS	Coarse grain. Uses 4 species of strings (nodes)	Fixed Hierarchy	ANN, Func. Opt., ...
GAME	Parallel version not available yet. Object Oriented	Any Desired	TSP, Func. Opt., ...
GDGA	Coarse Grain. Admits explicit exploration/exploitation	Hypercube	Func. Opt. (floating p.)
GENITOR II	Coarse grain. Interesting crossover operator	Ring	Func. Opt. and ANN's
HSDGA	Hierarchical coarse and fine grain GA. Uses E. S.	Ring, Tree, Star, ...	Function Optimization
PARAGENESIS	Coarse grain. Made for the CM-200 (1 ind. - 1 CPU)	Multiple	Function Optimization
PeGAsuS	Coarse or fine grain. High-level programming. MIMD	Multiple	Teaching and Func. Opt.
PGA 2.5	Coarse grain. Makes migrations among sub-populations	Multiple	Knapsack and Func. Opt.
PGAPack	Global parallelization (parallel evaluations)	Master / Slave	Function Optimization
RPL2	Coarse grain. Very flexible and open to new models	Any Desired	Research and Business
SGA-Cube	Coarse Grain. Made for the nCUBE 2	Hypercube	Function Optimization

5. TECHNICAL ISSUES FOR cgpGAs

In this section we want to present the most important advances in the procedures governing the evolution of a cgpGA. Most of them are related to the operators being applied. The rest are related to the parameters and techniques driving the migration between islands and the heterogeneity of the cgpGA.

5.1. The Evolution Mode

Relating the kind of evolution the cgpGA performs there exist basically two important models: the *generational* [26] and the *steady-state* [67] models. In fact they are the two extreme ends of the granularity of the evolution (algorithmic) step. The class of *messy-GAs* [28] is an exponent of a very different kind of search.

Traditionally any island creates a new population of individuals from the old one by applying genetic operators as selection, crossover and mutation. However this basic generational GA is somewhat slow and that it loses diversity quickly. The offspring is not available for use until the new generation.

On the other hand steady-state evolution creates in every step just a single new string by applying genetic operations, and inserts it back according to a given replacement policy (for example only if it is better than the worst existing string or randomly). This kind of GAs behaves very well in terms of efficacy, slower loss of diversity and very fast evolution (up to 10 times faster than as generational GA) in many real applications [20], [72], [41]. Parents and children coexist in the same population and we can use the new offspring immediately for further operations.

Messy GAs have very different genotypes that also highly influence the kind of needed genetic operations. They have also showed to be a fast class of algorithms.

A cgpGA with sub-populations making steady-state evolution is thought to perform better. In fact this is true and represents one of our present research lines that is working out excellent results.

5.2. The Selection Operator

The selection mechanism of a cgpGA plays an important role. It drives the search towards better individuals ensuring the convergence. This is directly related to the trade-off between high convergence velocity and high probability of finding a global optimum in complex problems. In this section we want to present a brief resume of the more important selection schemes used in evolutionary algorithms. In the following table λ is the number of strings in one population and $(\vec{a}_i(t))$ is a string at evolution step *t*.

TABLE IX.- SELECTION SCHEMES

Selection Scheme	Formalization	Comments
Proportional	$p_s(\vec{a}_i(t)) = f(\vec{a}_i(t)) \Big/ \sum_{j=1}^{\lambda} f(\vec{a}_j(t))$	[36] Called Roulette Wheel
Linear Ranking	$p_s(\vec{a}_i(t)) = \frac{1}{\lambda} \cdot \left(\eta_{max} - (\eta_{max} - \eta_{min}) \cdot \frac{i-1}{\lambda-1} \right)$	[12] $\eta_{min} = 2 - \eta_{max}$ and $1 \le \eta_{max} \le 2$. Individuals are sorted by fitness from 1 to λ
Whitleys's Linear Ranking	$Index(\chi, b) = \frac{\lambda}{2 \cdot (b-1)} \cdot \left(b - \sqrt{b^2 - 4 \cdot (b-1) \cdot \chi} \right)$	[72] It returns the index of the selected individual. $b \in [1,2]$ is the selection pressure (bias) and $\chi \in [0,1]$ is a random unif. variable
(μ,λ)-Uniform Ranking	$p_s(\vec{a}_i(t)) = \begin{cases} 1/\mu & 1 \le i \le \mu \\ 0 & \mu < i \le \lambda \end{cases}$	[62] Typical in Evolution Strategies. Individuals are sorted by fitness from 1 to λ

The proportional selection uses the fitness by itself to select individuals but the ranked mechanisms use the position of the string in a sorted population (from 1 to λ) with increasing values of fitness (we suppose a maximization problem without any loss of generality). This is thought to help the search when fitness values are very similar (for example at the end of the evolution).

We classify any selection scheme relating a number of different criteria (table X). All these criteria are discussed by showing the probability of selection of any individual. In any case the condition $\sum_{i=1}^{\lambda} p_s(\vec{a}_i(t)) = 1$ holds. The selection pressure of the algorithms toward fittest strings can be controlled by choosing the appropriate values of the selection mechanism. For example for linear ranking we can choose different constants. Also most of selection procedures admit versions and combinations.

TABLE X.- POSSIBLE VERSIONS OF THE SELECTION SCHEMES

Attending to...	Selection Scheme Version 1 (YES)	Selection Scheme Version 2 (NO)
Do we change the probabilities across gens.?	*Dynamic* $\Leftrightarrow \exists i \in \{1,\ldots,\lambda\}, \forall t \geq 0 \mid p_s(\vec{a}_i(t)) = c_i$	*Static* $\Leftrightarrow \forall i \in \{1,\ldots,\lambda\}, \forall t \geq 0 \mid p_s(\vec{a}_i(t)) = c_i$
Is it allowed to appear non-zero probabilities?	*Extinctive* $\Leftrightarrow \forall t \geq 0, \exists i \in \{1,\ldots,\lambda\} \mid p_s(\vec{a}_i(t)) = 0$	*Preservative* $\Leftrightarrow \forall i \in \{1,\ldots,\lambda\}, \forall t \geq 0 \mid p_s(\vec{a}_i(t)) > 0$
Do Worst individuals reproduce?	*Left Extinctive* $\Leftrightarrow \forall t \geq 0, \exists l \in \{1,\ldots,\lambda-1\} \mid i \leq l \Rightarrow p_s(\vec{a}_i(t)) = 0$	*Right Extinctive* $\Leftrightarrow \forall t \geq 0, \exists l \in \{2,\ldots,\lambda\} \mid i \geq l \Rightarrow p_s(\vec{a}_i(t)) = 0$
Shall best k parents undergo selection with their offspring?	*k-Elitist* $\Leftrightarrow \exists k \in \{1,\ldots,\lambda\}, \forall t \geq 0, \forall i \in \{1,\ldots,k\} \mid f(\vec{a}_i(t)) \geq f(\vec{a}_i(t-1))$	*Pure* <No individual satisfies the *k-Elitist* property>
Shall generate a whole population of λ individuals?	*Generational* <Fixed set of parents until λ offspring>	*Steady-State* <Special case of (λ-1)-elitism>

Any node in the cgpGA is thought to apply one of these selection algorithms. Other interesting operators like *tournament* or *stochastic universal sampling* can be found elsewhere [26], [13], [10].

5.3. The Replacement Policy

The replacement policy is used in each node of the cgpGA in order to replace some individuals in the present population with a new individual(s). The following table shows the probabilities of replacement of several popular policies.

TABLE XI.- DIFFERENT POLICIES FOR THE REPLACEMENT OF STRINGS

Replacement Scheme	Formalization	Comments
Inverse Proportional	$p_r(\vec{a}_i(t)) = 1 - f(\vec{a}_i(t)) \Big/ \sum_{j=1}^{\lambda} f(\vec{a}_j(t))$	The inverse of the roulette wheel selection
Uniform Random	$p_r(\vec{a}_i(t)) = c = \frac{1}{\lambda}$	Any individual can be replaced attending to the same probability (typical in steady-state)
Worst	$p_r(\vec{a}_i(t)) = \begin{Bmatrix} 1 & i = \lambda \\ 0 & i \neq \lambda \end{Bmatrix}$	Always replace the worst string in a sorted population. A popular version replaces the worst string only if the new one is better
Generational	$p_r(\vec{a}_i(t)) = 1$	Canonical generational GAs replace the whole old population with a new one

Just like the selection policy the replacement admits different versions and hybridizations. Hence we can found an elitist generational replacement, and extinctive replacement, etc...

5.4. The Crossover Operator

The crossover is one of the most studied operators in parallel and sequential GAs. A great deal of variants have been defined. All of them can be included in one of the following classes:

- **Pure** crossover operators: robust procedures that work on any application.
- **Hybrid** crossover operators: mixing of a pure crossover operator and some other algorithm.
- **Problem dependent** operators: they make very specialized operations on the strings, usually by including specific problem knowledge.

The traditional crossover operates on two parents to yield two new offspring. However there exist many version which only compute one new string from two parents. Also it is normal for combinatorial optimization or in ordering problems to use a safe crossover in that the offspring of two correct strings are also two new correct strings. This is an imposition in some problems when not all the possible strings are allowable solutions e.g. for managing restrictions. Often the restrictions are managed in the fitness function (if a string does not match a given problem restriction its fitness will be penalized severely). However some applications find more efficient to force the crossover to work out only correct strings [5]. A third solution could be repairing the incorrect strings.

The most traditional crossover defines randomly a position in every parent string as the crossing point. Afterwards the two strings exchange their slices located before and after the crossing point.

Multiple extensions to this single point crossover (SPX) are possible and are known as *N*-point crossover operators [63]. The most interesting one in this family is the two point crossover (DPX).

In DPX two crosspoints are defined and the two parents exchange the slices bounded by these two limits. An interesting version we call DPX1 has been successfully used to yield only one child that contains the largest possible slice from the best (higher fitness) parent [6].

❖ Single Point Crossover (SPX)

$c_{\{p_c\}}$**:GxG → GxG** $\quad \tau \in [0,1] \quad \alpha \in \{1,\ldots,l\}$

if $\tau > p_c$ **then** $c_{\{p_c\}}\left(\vec{a}=(a_1,\ldots,a_l),\vec{b}=(b_1,\ldots,b_l)\right)=\left((a_1,\ldots,a_\alpha,b_{\alpha+1},\ldots,b_l),(b_1,\ldots,b_\alpha,a_{\alpha+1},\ldots,a_l)\right)$

else $c_{\{p_c\}}\left(\vec{a}=(a_1,\ldots,a_l),\vec{b}=(b_1,\ldots,b_l)\right)=\left((a_1,\ldots,a_l)(b_1,\ldots,b_l)\right)$

❖ Double Point Crossover 1-child (DPX1)

$c_{\{p_c\}}$**:GxG → G** $\quad \tau \in [0,1] \quad \alpha,\sigma \in \{1,\ldots,l\},\sigma \geq \alpha \quad \Phi(\vec{a}) \geq \Phi(\vec{b}) \wedge (\sigma-\alpha) < l/2$

if $\tau > p_c$ **then** $c_{\{p_c\}}\left(\vec{a}=(a_1,\ldots,a_l),\vec{b}=(b_1,\ldots,b_l)\right):=(a_1,\ldots,a_\alpha,b_{\alpha+1},\ldots,b_\sigma,a_{\sigma+1},\ldots,a_l)$

else $c_{\{p_c\}}\left(\vec{a}=(a_1,\ldots,a_l),\vec{b}=(b_1,\ldots,b_l)\right)=(a_1,\ldots,a_l)$

❖ Biased Uniform Crossover (UX(P_b))

$c_{\{p_c\}}$**:GxG → GxG** $\quad \tau \in [0,1]\ \alpha_i \in [0,1] \quad \Phi(\vec{a}) \geq \Phi(\vec{b})$ usually $P_b \geq 0.5$ $\quad \alpha_i$ sampled anew for every allele

if $\tau > p_c$ **then** $c_{\{p_c\}}(\vec{a},\vec{b})=\left((\vec{a}',\vec{b}')\right), a_i^{\prime}=\begin{Bmatrix} a_i & \alpha_i > P_b \\ b_i & \alpha_i \leq P_b \end{Bmatrix},\ b_i^{\prime}=\begin{Bmatrix} b_i & \alpha_i > P_b \\ a_i & \alpha_i \leq P_b \end{Bmatrix} \quad \forall i \in \{1,\ldots,l\}$

else $c_{\{p_c\}}\left(\vec{a}=(a_1,\ldots,a_l),\vec{b}=(b_1,\ldots,b_l)\right)=\left((a_1,\ldots,a_l)(b_1,\ldots,b_l)\right)$

❖ Biased Arithmetic Crossover (AX(P_b))

$c_{\{p_c\}}$**:GxG → GxG** $\quad \tau \in [0,1] \quad \Phi(\vec{a}) \geq \Phi(\vec{b})$ $\quad$ usually $P_b \geq 0.5$

if $\tau > p_c$ **then** $c_{\{p_c\}}(\vec{a},\vec{b})=\left((\vec{a}',\vec{b}')\right), a_i^{\prime}=P_b\cdot a_i+(1-P_b)\cdot b_i,\ b_i^{\prime}=P_b\cdot b_i+(1-P_b)\cdot a_i \quad \forall i \in \{1,\ldots,l\}$

else $c_{\{p_c\}}\left(\vec{a}=(a_1,\ldots,a_l),\vec{b}=(b_1,\ldots,b_l)\right)=\left((a_1,\ldots,a_l)(b_1,\ldots,b_l)\right)$

The uniform crossover (UX) is usually implemented by creating two offspring with every allele in the new offspring taken randomly from one parent (P_b=0.5). Here we present a generalized biased version in which the alleles of the best parent have higher probability of appearing in the child.

The arithmetic crossover (AX) has the same aspect but every child allele is a weighted new value of the two parental alleles. It is intended to be useful for floating point codes.

These two operators (UX and AX) are independent of the length of the string. This characteristic is useful in many real world applications.

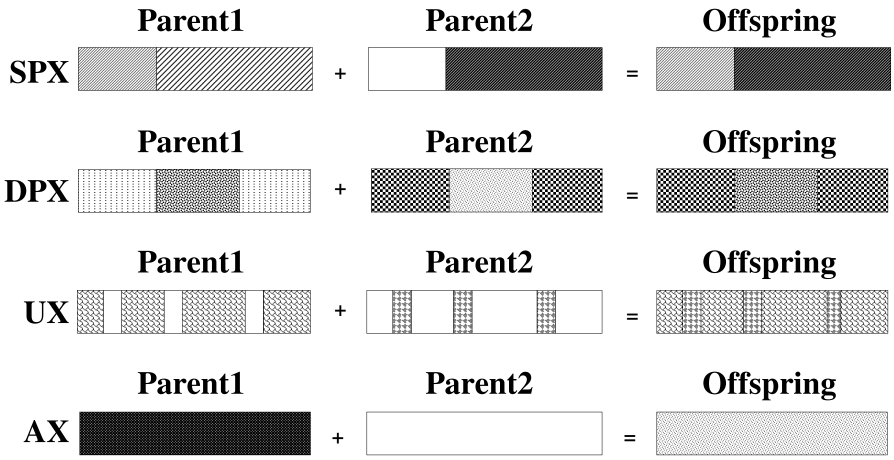

Figure 9.- W*ork ofseveral crossover operators (only one child is showed).*

In general there exist typical operators with special efficacy depending on the field of application. Thus for training neural networks we can use a traditional DPX or else a reduced surrogate operator [72]. For the traveling salesperson problem (TSP) we can use circuit crossovers as Partially Matched Crossover (PMX), Cyclic Crossover (CX), Order Crossover (OX), etc...[49]. Also the interesting and recent incorporation of fuzzy connectives to develop new classes of crossover operators is very promising in particular for real codes (see [34] for more details).

5.5. The Mutation Operator

The mutation operator is intended to introduce new genotypes in the genetic pool of strings. The goal is to maintain a minimum diversity that allows a continuous search towards a solution. This is needed to scape from local optima when selection and crossover have led to slightly bad regions of exploration. Also it can help in refining a sub-optimal solution. In a cgpGA the migration of individuals from a neighbor island also extends the exploration in the distributed populations (new genotypes).

❖ **Bit-Flip Mutation**

$$m_{\{p_m\}}:\mathbf{G}\rightarrow\mathbf{G} \quad m_{\{p_m\}}(s_1,\ldots,s_l)=(s'_1,\ldots,s'_l) \quad \forall j\in\{1,\ldots,l\} \quad s'_j=\begin{cases} s_j & \alpha_i>p_m \\ 1-s_j & \alpha_i\le p_m \end{cases} \quad \alpha_j\in[0,1]$$

The traditional mutation works on binary strings by probabilistically changing every position (allele) to its complementary value. Also many possible versions exist. For example for real codes the mutation usually adds (subtracts) some random value to the allele.

An interesting version of any mutation operator consists in varying its probability throughout the evolution. Hence we could increase very slightly the mutation probability in every iteration. This is adequate since at the beginning the genotypes are different enough to sustain exploration, but at the end they get very similar. This **dynamic** probability can follow an *a priori given linear or exponential* growth or else be computed for example as the *inverse of the hamming distance* of the strings in the population. See [39] for more details. This dynamic variant is also possible for crossover or else for other hybrid operators we would want to apply (if guided by a probability of application).

5.6. Migration

In relation with the migration many aspects have to be precised in order to define the kind of cgpGA we are dealing with. In this section we want to outline the most accepted techniques.

5.6.1. Migration Gap

Since a cgpGA usually makes sparse exchanges of individuals among the sub-populations we must define the migration gap, i. e., the number of steps in every sub-population between two successive exchanges. This can be made in every sub-population periodically or else by using a given probability of migration P_M to decide in every step whether migration will take place or not. A different approach can be found for example in [52] in which the authors explore the *sigma-exchange algorithm* consisting in only making migrations when the standard deviation is small enough to make sense the exchange of strings. Although this criterium can reduce the number of exchanges it relies on the fitness and not on the genotypes of the parallel islands (this could be a disadvantage for certain problems).

It is very usual for a cgpGA to be **synchronous** in that the phases of migration and reception of external migrants are tighted in the same portion of the algorithm. However synchronous migrations are known to be very slow [52], [70]. In **asynchronous** cgpGAs the reception of individuals are scheduled to occur whenever during a run of every island in the cgpGA. This means sending migrants whenever it is needed and also accepting migrants whenever they arrive from a neighbor population.

This last asynchronous cgpGA can perform very efficiently. Besides that its behavior and theoretical study will be similar to synchronous cgpGAs. However if the sub-algorithms run in very different processors the exchanged individuals can be in very different stages of evolution. Thus the well known **non-effect** problem (the incoming migrant is very bad in its new population) or else the **super-individual** problem (the incoming migrant is much better than the population receiving it) can appear.

5.6.2. Migration Rate

The migration rate is a parameter determining the number of individuals that undergo migration in every exchange. This value is expressed as a percentage of the population or else as an absolute value. Anyway it is not clear which is the best value for the migration rate. It seems to be clear that a low percentage should be used (1%..10% of the population) but, although many studies have been conducted, no clear conclusions can be drawn. In fact it seems that, for some problems, sending 1 individual is better than sending 5 individuals, and viceversa for other different problems.

We think this is due to the close correlation of this value with the migration gap and topology. Similar results can be reached by different set of values of these three parameters. Therefore if we use constant values of 1 individual for the migration rate and a ring topology then we can get an almost universal cgpGA. Then the migration gap can control the behavior of the whole algorithm to get it near/far the advantages/drawbacks of other different cgpGAs: <*topo*=`ring`, *mr*=`1`, *mig_gap*=`?`>.

5.6.3. Selection and Replacement in the Migration Procedure

It is very common in cgpGAs that the same selection and replacement algorithms of the islands are used for dealing with migrants. Hence all the presented models in sections 5.2 and 5.3 are useful here.

Two distinguished selection procedures for the migrants are both sending the **best** individual or sending a **random** one. In fact if you have no a priori knowledge on the problem under consideration you might decide to migrate a random individual. This is so since it is safest for a large range of migration gaps (that you may also be trying to set correctly). If every sub-algorithm is sending the best individual then the migration gap needs to be considerably larger in order to avoid the premature convergence of the whole population of the cgpGA.

5.6.4. Topology

There exist plenty of works trying to decide the best topology for a cgpGA [1], [15], [31], [44]. It seems that for the most of problems the ring and hypercube are two of the best topologies. Full-connected and centralized topologies have problems in cgpGAs due the tight connectivity. In fact we believe the ring topology is the best in terms of the quality of the search and also because it is very simple. This means easy to implement on very different hardware platforms (cluster of workstations, multicomputers, ...).

It is very usual for the topologies to be unidirectional. Hierarchical topologies as trees of sub-algorithms have also been successfully applied [70]. A hierarchical topology is a natural way of making the search at different levels. For example in every level you can use a GA with a different kind of genotype and then get a cgpGA that first roughly determines the promising zones (leaves) and then in higher levels the cgpGA is exploiting more accurate solutions inside these zones [56].

The traditional nomenclature divides cgpGAs into **stepping-stone** or **island** models depending on whether the individuals can freely migrate to any sub-population or they are restricted to migrate to geographically nearby islands, respectively. Most recent papers assume to work with a given topology.

5.7. Heterogeneity

The homo/hetero-geneity of a distributed system is usually understood as a term referring to the hardware. This is also correct for a cgpGA since it runs in several machines. However we have an additional level for possible heterogeneity relating the kind of search the islands are making (fig. 3).

The *d* sub-algorithms can be performing the same kind of search (same genotypes, operators, ...) on different sets of randomly generated individuals or else the sub-algorithms can be using different sets of operators or parameters or genotypes. These last heterogeneous systems can have some advantages on homogeneous cgpGAs since they can be explicitly tuned to make exploration and exploitation (a well-known trade-off decision if evolutionary algorithms) depending on the problem.

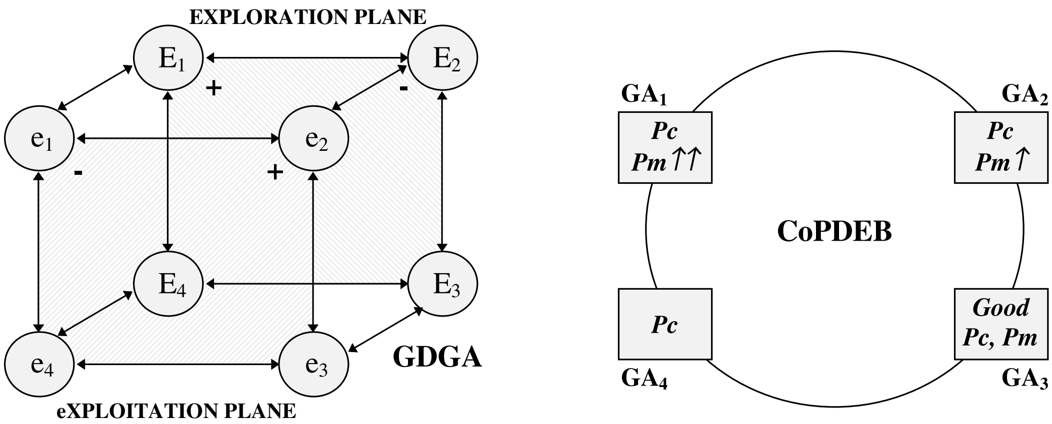

Figure 10.- *Two recent cgpGAs: GDGA (left) and CoPDEB (right).*

The Gradually Distributed GA (GDGA) [45] creates two planes of search with exploitation and exploration properties embedded in several fuzzy crossover operators (on real codes). The backward and forward migrations exchange individuals between these two planes of the hypercube-3. It is a synchronous model based in generational evolution. The Co-operating Populations with Different Evolution Behavior (CoPDEB) [2] on the other hand applies traditional operators but it changes the probabilities of crossover and mutation in every sub-population in order to get different GAs with also different exploration/exploitation properties. Besides that, local hill-climbing is included in islands.

6. IMPLEMENTATION ISSUES

Among all the languages for developing cgpGAs, `C` has been widely used for these systems. However implementations with `C++` are now growing in number due to the advantages in software reusing, security and extensibility. Also, because of its large capability of structured parameterization, an object oriented language has advantages in composing new prototypes. On the other hand there exist many cgpGAs developed for concrete machine architectures that use embedded languages (versions of `C`, ...).

The communication among processes is normally achieved by using the *BSD socket interface* on UNIX systems, either directly or through the services of the well known Parallel Virtual Machine (*PVM*) [25]. Some *MPI* [43] and `JAVA` implementations are becoming familiar. Finally many systems simulate the parallelism in a single process. This last is useful only when the basic behavior is to be studied. However for real or complex applications a truly parallel cgpGA is needed.

7. CONCLUSIONS

In this work we have presented the fundamental syntax and semantics of sequential genetic algorithms (GAs). The paper deals with a very popular extension of these algorithms known as coarse grain parallel GA (cgpGA) in which many sub-algorithms run in parallel with sparse migrations of strings. The robustness and advantages of sequential GAs are enhanced when a cgpGA is used. Its only drawback is the more complex design and the need of a parallel (distributed) machine to run it.

An structured overview on the more important cgpGA systems is discussed and useful recent information about them is also outlined. We included theoretical foundations and descriptions of a cgpGA as well as many possible variants apart from its basic operation. We are specially worried about offering useful and rigorous material that could help new or old practitioners. Although our overview is obviously not complete it represents a good start point to conduct future research on this domain or to make new applications by using coarse grain parallel GAs.

References

[1] P. Adamidis. “Review of Parallel Genetic Algorithms Bibliography”. *Internal T.R., Aristotle University of Thessaloniki*, November (http://www.control,ee.auth.gr/~panos/papers/pga_review.ps.gz). 1994.

[2] P. Adamidis & V. Petridis. “Co-operating Populations with Different Evolution Behavior”. *Proceedings of the Second IEEE Conference on Evolutionary Computation*, pp 188-191. 1996.

[3] E. Alba, J. F. Aldana & J. M. Troya. “Load Balancing and Query Optimization in Dataflow Parallel Evaluation of Datalog Programs”. *Proceedings of the International Conference on Parallel and Distributed Systems*, Lionel M. Ni (ed.). IEEE Computer Society Press, pp 682-688. 1994.

[4] E. Alba, J. F. Aldana & J. M. Troya. “A Genetic Algorithm for Load Balancing in Parallel Query Evaluation for Deductive Relational Databases”. *Procs. of the I. C. on ANNs and GAs*, D.W. Pearson, N.C. Steele, R.F. Albrecht (eds.), pp 479-482. Springer-Verlag. 1995.

[5] E. Alba, C. Cotta & J. M. Troya. “Type-Constrained Genetic Programming for Rule-Base Definition in Fuzzy Logic Controllers”. *Proceedings of the First Annual Conference on Genetic Programming*, J. R. Koza, D. E. Goldberg, D. B. Fogel & R. L. Riolo (eds.), Stanford Univ., Cambridge, MA. The MIT Press, pp 255-260. 1996.

[6] E. Alba & J. M. Troya. “Genetic Algorithms for Protocol Validation”. *Proceedings of the PPSN IV I.C.*, H. M. Voigt, W. Ebeling, I. Rechenberg & H. P. Schwefel (eds.), Berlin. Springer-Verlag, pp 870-879. 1996.

[7] E. Alba & C. Cotta. “Evolution of Complex Data Structures”. *To appear in the AEIA Journal*, No. ?. 1997.

[8] C. Alippi & P. Treleaven. “GAME: A Genetic Algorithms Manipulation Environment”. *Internal Report Department of Computer Science*, UCL. 1991.

[9] J. Antonisse. "A New Interpretation of Schema Notion that Overturns the Binary Encoding Constraint". *Proceedings of the 3rd ICGA*, J. D. Schaffer (ed.), Morgan Kaufmann, pp 86-91. 1989.

[10] T. Bäck & F. Hoffmeister. "Extended Selection Mechanisms in Genetic Algorithms". *Proceedings of the Fourth International Conference on Genetic Algorithms*, R. K. Belew and L. B. Booker (eds.), Morgan Kaufmann, San Mateo, CA, pp 92-99. 1991.

[11] T. Bäck & H-P Schwefel. "An Overview of Evolutionary Algorithms for Parameter Optimization". *Evolutionary Computation*, 1 (1), pp 1-23, The MIT Press. 1993.

[12] J. E. Baker. "Adaptive Selection Methods for Genetic Algorithms". *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, J. J. Grefenstette (ed.), Lawrence Erlbaum Associates, Publishers, pp 101-111. 1987.

[13] J. E. Baker. "Reducing Bias and Inefficiency in the Selection Algorithm". *Proceedings of the Second International Conference on Genetic Algorithms*, J. J. Grefenstette (ed.), Lawrence Erlbaum Associates, Publishers, pp 14-21. 1987.

[14] T. C. Belding. "The Distributed Genetic Algorithm Revisited". *Proceedings of the 6th International Conference on GAs*, L. J. Eshelman (ed.), Morgan Kaufmann, pp 122-129. 1995.

[15] E. Cantú-Paz. "Algoritmos Genéticos Paralelos en Arquitecturas de Memoria Distribuida". *Ph. D. Thesis Dissertation*, Instituto Tecnológico Autónomo de México. May 1994.

[16] E. Cantú-Paz. "A Summary of Research on Parallel Genetic Algorithms". *R. 95007*, July 1995. Also revised version, *IlliGAL R. 97003*. May 1997.

[17] A. Chipperfield & P. Fleming. "Parallel Genetic Algorithms". *Parallel and Distributed Computing Handbook*, A. Y. H. Zomaya (ed.), MacGraw-Hill , pp 1118-1143. 1996.

[18] A. L. Corcoran & R. L. Wainwright. "LibGA: A User-Friendly Workbench for Order-Based Genetic Algorithm Research". *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied*

Computing, ACM Press. 1993.

[19] F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza-Reorda. "Exploiting Competing Subpopulations for Automatic Generation of Test Sequences for Digital Circuits". *Procs. of the PPSN IV I.C.*, H. M. Voigt, W. Ebeling, I. Rechenberg & H. P. Schwefel (eds.), Springer-Verlag, pp 792-800. 1996.

[20] C. Cotta, E. Alba & J. M. Troya. "Evolutionary Design of Fuzzy Logic Controllers". IEEE Catalog N. 96CH35855, *Proceedings of the ISISC'96 Conference*, Dearborn, MI pp 127-132. 1996.

[21] Y. Davidor. "A Naturally Occurring Niche & Species Phenomenon: The Model and First Results". *Procs. of the 4th ICGA*, R. K Belew & L. B. Booker (eds.), pp 257-263. 1991.

[22] L. Davis & J. J. Grefenstette. "Concerning GENESIS and OOGA". *Handbook of Genetic Algorithms,* L. Davis (ed.), New York: Van Nostrand Reinhold, pp 374-376. 1991.

[23] J. A. Erickson, R. E. Smith, D. E. Goldberg. "SGA-Cube, A Simple Genetic Algorithm for nCUBE 2 Hypercube Parallel Computers". *TCGA Report No. 91005*, The Univ. of Alabama. 1991.

[24] L. J. Fogel, A. J. Owens, M. J. Walsh. *Artificial Intelligence Through Simulated Evolution*. John Wiley, New York. 1966.

[25] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam. *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. The MIT Press. 1994.

[26] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*; Addison-Wesley. 1989.

[27] Goldberg, D. E.. "Sizing Populations for Serial and Parallel Genetic Algorithms". *Proceedings of the 3rd ICGA*, J. D. Schaffer (ed.), Morgan Kaufmann, pp 70-79. 1989.

[28] D. E. Goldberg, K. Deb & B. Korb. "Don't Worry, Be Messy". *Proceedings of the Fourth International Conference on Genetic Algorithms*, R. K. Belew and L. B. Booker (eds.), Morgan Kaufmann, San Mateo, CA, pp 24-30. 1991

[29] Goldberg, D. E. et al. "Critical Deme Size for Serial and Parallel Genetic Algorithms". *IlliGAL Report No. 95002*. January 1995.

[30] E. D. Goodman. An Introduction to GALOPPS v3.2. *TR#96-07-01*, GARAGe, I. S. Lab., Dpt. of C. S. and C. C. C. A. E. M., Michigan State University, East Lansing. 1996.

[31] V. S. Gordon and D. Whitley. "Serial and Parallel Genetic Algorithms as Function Optimizers". *Procs. of the 5th ICGA*, S. Forrest (ed.), Morgan Kaufmann, pp 177-183. 1993.

[32] M.Gorges-Schleuter. "ASPARAGOS An Asynchronous Parallel Genetic Optimisation Strategy". *Procs. of the 3rd ICGA*, J. D. Schaffer (ed.), Morgan Kaufmann, pp. 422-427. 1989.

[33] J. J. Grefenstette. "GENESIS: A System for Using Genetic Search Procedures". *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, pp. 161-165. 1984.

[34] F. Herrera, M. Lozano, J. L. Verdegay. "Tackling Fuzzy Genetic Algorithms". *Genetic Algorithms in Engineering and Computer Science*, G. Winter, J. Périaux, M. Galán, P. Cuesta (eds.), John Wiley & Sons, pp 167-189. 1995.

[35] F. Hoffmeister. "The User's Guide to ESC^{A}P$_{A}$DE 1.2 A Runtime Environment for Evolution Strategies". *Department of Computer Science*, University of Dortmund, Germany. 1991.

[36] J. H. Holland. *Adaptation in Natural and Artificial Systems*. Univ. of Michigan Press, Ann Arbor, MI. 1975.

[37] M. Hughes. "Genetic Algorithm Workbench Documentation", Cambridge Consultants Ltd. 1989.

[38] A. Hunter. "Sugal Programming Manual v2.0". *T. R.* in Univ. Sunderland, U.K.. July 1995.

[39] B. A. Juslstrom. "What Have You Done for Me Lately? Adapting Operator Probabilities in a Steady-State Genetic Algorithm". *Proceedings of the 6th International Conference on Genetic Algorithms*, L. J. Eshelman (ed.), Morgan Kaufmann, pp 81-87. 1995.

[40] J. R. Koza. *Genetic Programming*. The MIT Press. 1992.

[41] T. Kuo & S. Y. Hwang. “A Genetic Algorithm with Disruptive Selection”. *Proceedings of the 5th International Conference on GAs*, S. Forrest (ed.), Morgan Kaufmann, pp 65-69. 1993.

[42] D. Levine. “A Parallel Genetic Algorithm fot the Set Partitioning Problem”. *T. R. No. ANL-94/23*, Argonne National Laboratory, Mathematics and Computer Science Division. 1994.

[43] D. Levine. “Users Guide to the PGAPack Parallel Genetic Algorithm Library”. *T. R. ANL-95/18*, January 31. 1996.

[44] S. Lin, W. F. Punch and E. D. Goodman. “Coarse-Grain Parallel Genetic Algorithms: Categorization and New Approach”. *Parallel & Distributed Processing*. October 1994.

[45] M. Lozano. “Application of Fuzzy Logic Based Techniques for Improving the Behavior of GAs with Floating Point Encoding”. *Ph.D. Th., Dpt. of C. S. and A.I.*, Univ. Granada. July 1996.

[46] B. Manderick and P. Spiessens. “Fine-Grained Parallel Genetic Algorithms”. *Procs. of the 3rd I. C. on Genetic Algorithms*, J. D. Schaffer (ed.), Morgan Kaufmann, pp 428- 433. 1989.

[47] M. Mejía-Olvera & E. Cantú-Paz. “DGENESIS-Software para la Ejecución de Algoritmos Genéticos Distribuidos”. *Proceedings of the XX Conferencia Latinoamericana de Informática,* pp 935-946, Monterrey, México. 1994.

[48] J. J. Merelo & A. Prieto. GAGS, “A Flexible Object-Oriented Library for Evolutionary Computation”. *Procs. of MALFO*, D. Borrajo & P. Isasi (eds.), pp 99-105. 1996.

[49] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag. 1992.

[50] H. Mühlenbein. “Evolution in Time and Space - The Parallel Genetic Algorithm”. *Foundations of Genetic Algorithms*, G. J. E. Rawlins (ed.), Morgan Kaufmann, pp 316-337. 1991.

[51] H. Mühlenbein, M. Schomisch & J. Born. “The Parallel Genetic Algorithm as Function Optimizer”. *Parallel Computing*, 17, pp 619-632. September 1991.

[52] M. Munetomo, Y. Takai & Y Sato. "An Efficient Migration Scheme for Subpopulation-Based Asynchronously Parallel GAs". *HIER-IS-9301*, Hokkaido University. July 1993.

[53] T. Muntean & E. G. Talbi. "A Parallel Genetic Algorithm for Process-Processors Mapping". *Proceedings of the Second Symposium II. High Performance Computing*, M. Durán & E. Dabaghi (eds.), pp 71-82. Montpellier, France: F. Amsterdam, Amsterdam. 1991.

[54] NASA - Johnson Space Center. "Splicer - A Genetic Tool for Search and Optimization". *Genetic Algorithm Digest, Vol 5, Issue 17*. 1991.

[55] C. C. Pettey, M. R. Leuze & J. Grefenstette. "A Parallel Genetic Algorithm". *Proceedings of the 2nd ICGA*, J. Grefenstette (ed.), Lawrence Erlbraum Associates, pp 155-161. 1987.

[56] J. C. Potts, T. D. Giddens & S. B. Yadav. "The Development and Evaluation of an Improved Genetic Algorithm Based on Migration and Artificial Selection". *IEEE Transactions on Systems, Man, and Cybernetics*, 24 (1), pp 73-86. January 1994.

[57] N. J. Radcliffe & P. D. Surry. "The Reproductive Plan Language RPL2: Motivation, Architecture and Applications". *Genetic Algorithms in Optimisation, Simulation and Modelling*, J. Stender, E. Hillebrand & J. Kingdon (eds.), IOS Press. 1994

[58] I. Rechenberg. *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien der Biologischen Evolution*. Fromman-Holzboog Verlag, Sttutg. 1973.

[59] J. L. Ribeiro Filho, C. Alippi & P. Treleaven. "Genetic Algorithm Programming Environments". *Parallel Genetic Algorithms: Theory & Applications*, J. Stender (ed.), IOS Press. 1993.

[60] G. Robbins. "EnGENEer - The Evolution of Solutions". *Proceedings of the 5th Annual Seminar on Neural Networks and Genetic Algorithms*. 1992.

[61] N. N. Schraudolph & J. J. Grefenstette. "A User's Guide to GAUCSD 1.2". *T. R. Computer Science & Engineering Department*, University of California, San Diego. 1991.

[62] H. P. Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chichester. 1981.

[63] W. M. Spears & K. A. De Jong. "An Analysis of Multi-Point Crossover". *Foundations of Genetic Algorithms*, G. J. E. Rawlins (ed.), Morgan Kaufmann, pp 301-315. 1991.

[64] W. M. Spears. "Crossover or Mutation?". *Proceedings of the Foundations of Genetic Algorithms Workshop*, D. Whitley (ed.), Morgan Kaufmann, pp 221-237. 1992.

[65] P. Spiessens and B. Manderick. "A Massively Parallel Genetic Algorithm". *Proceedings of the 4th International Conference on Genetic Algorithms*, R. K. Belew and L. B. Booker (eds.), Morgan Kaufmann, pp 279-286. 1991.

[66] J. Stender (ed.). *Parallel Genetic Algorithms: Theory and Applications*. IOS Press. 1993.

[67] G. Syswerda. "A Study of Reproduction in Generational and Steady-State Genetic Algorithms"; *Foundations of GAs*, G. J. E. Rawlins (ed.), Morgan Kaufmann, pp 94-101. 1991.

[68] R. Tanese. "Distributed Genetic Algorithms". *Proceedings of the 3rd International Conference on Genetic Algorithms*, J. D. Schaffer (ed.), Morgan Kaufmann, pp 434-439. 1989.

[69] H. M. Voigt, J. Born & J. Treptow. "The Evolution Machine Manual - V 2.1". *T. R.* in the Institute for Informatics and Computing Techniques, Berlin. 1991.

[70] H. M. Voigt, I. Santibáñez-Koref & J. Born. "Hierarchically Structured Distributed Genetic Algorithms". *Proceedings of the International Conference Parallel Problem Solving from Nature, 2*, R. Männer and B. Manderick (eds.), North-Holland, Amsterdan, pp 155-164. 1992.

[71] M.Wall. "Overview of Mattew's Genetic Algorithm Library". http://lancet.mit.edu/ga. 1995.

[72] D. Whitley. "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best". *Proceedings of the 3rd. International Conference on Genetic Algorithms*, J. D. Schaffer (ed.), Morgan Kaufmann, pp 116-121. 1989.

[73] D. Whitley & T. Satarkweather. "GENITOR II: a Distributed Genetic Algorithm". J. Expt. theor.

Artif. Intelligence 2, pp 189-214. 1990.