|  |  |
| --- | --- |
| **I.** | **INTRODUCTION TO NATURE-INSPIRED ALGORITHMIC TECHNIQUES** |

**Contents of this chapter**

**I.1** Introduction to Evolutionary Computation

**I.1.1** Biological Basis

**I.1.2** The Skeleton of an Evolutionary Algorithm

**I.2** Important Paradigms in Evolutionary Computation

**I.3** Characterization of Evolutionary Computation

**I.4** Common Components in EC Algorithms

**I.5** A Distinguished EC Representative: Genetic Algorithms

**I.6** Relationship with other Techniques

**I.7** Important and Present Research Areas in Evolutionary Computation

**I.1 Introduction to Evolutionary Computation**

The term *evolutionary computation* refers to the study of the foundations and applications of certain heuristic techniques based on the principles of natural evolution. In spite of the fact that these techniques can be classified into three main categories (we call it "The Evolutionary Equation", see Figure 1), this classification is based in some details and historical development facts rather than in major functioning differences. In fact, their biological basis is essentially the same.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **EC** | **=** | **GA** | **+** | **ES** | **+** | **EP** |
| ***Evolutionary Computing*** |  | ***Genetic Algorithms*** |  | ***Evolution Strategies*** |  | ***Evolutionary Programming*** |
|  |  | [(Holland, 1975)](http://www.lcc.uma.es/~ccottap/semEC/Apprefs/apprefs.html#HO75) |  | [(Rechenberg, 1973)](http://www.lcc.uma.es/~ccottap/semEC/Apprefs/apprefs.html#RE73) |  | [(Fogel, Owens, Walsh, 1966)](http://www.lcc.uma.es/~ccottap/semEC/Apprefs/apprefs.html#FO66) |

**Figure 1.** *The Evolutionary Equation.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SC** | **=** | **EC** | **+** | **ANN** | **+** | **FL** |
| ***Soft Computing*** |  | ***Evolutionary Computation*** |  | ***Artificial Neural Networks*** |  | ***Fuzzy Logic*** |

**Figure 2.** *The Soft Computing Equation.*

**I.1.1 *Biological Basis***

The origin of evolutionary algorithms was an attempt to mimic some of the processes taking place in natural evolution. Although the details of biological evolution are not completely understood (even nowadays), there exist some points supported by a strong experimental evidence:

Evolution is a process operating over chromosomes rather than over organisms. The former are organic tools encoding the structure of a living being, i.e, a creature is "built" decoding a set of chromosomes.

Natural selection is the mechanism that relates chromosomes with the efficiency of the entity they represent, thus allowing those efficient organism which are well-adapted to the environment to reproduce more often than those which are not.

The evolutionary process takes place during the reproduction stage. There exists a large number of reproductive mechanisms in Nature. Most common ones are mutation (that causes the chromosomes of offspring to be different to those of the parents) and recombination (that combines the chromosomes of the parents to produce the offspring).

Based upon the features above, the three mentioned models of evolutionary computing were independently (and almost simultaneously) developed.

**I.1.2 *The Skeleton of an Evolutionary Algorithm***

An Evolutionary Algorithm (EA) is an iterative and stochastic process that operates on a set of individuals (population). Each individual represents a potential solution to the problem being solved. This solution is obtained by means of a encoding/decoding mechanism. Initially, the population is randomly generated (perhaps with the help of a construction heuristic). Every individual in the population is assigned, by means of a fitness function, a measure of its goodness with respect to the problem under consideration. This value is the quantitative information the algorithm uses to guide the search. The whole process is sketched in [Figure 3](http://www.lcc.uma.es/~ccottap/semEC/cap01/cap_1.html#Fi3).

|  |
| --- |
| *Generate* [P(0)]*t* http://www.lcc.uma.es/~ccottap/symbols/arrow_l.gif0**WHILE NOT** *Termination\_Criterion* [P(*t*)] **DO** *Evaluate* [P(*t*)]P' (*t*) http://www.lcc.uma.es/~ccottap/symbols/arrow_l.gif*Select* [P(*t*)]P''(*t*) http://www.lcc.uma.es/~ccottap/symbols/arrow_l.gif*Apply\_Reproduction\_Operators* [P'(*t*)]P(*t*+1) http://www.lcc.uma.es/~ccottap/symbols/arrow_l.gif*Replace* [P(*t*), P''(*t*)]*t* http://www.lcc.uma.es/~ccottap/symbols/arrow_l.gif*t* + 1 **ENDRETURN** *Best\_Solution* |

**Figure 3**. *Skeleton of an Evolutionary Algorithm.*

It can be seen that the algorithm comprises three major stages: selection, reproduction and replacement. During the selection stage, a temporary population is created in which the fittest individuals (those corresponding to the best solutions contained in the population) have a higher number of instances than those less fit (natural selection). The reproductive operators are applied to the individuals in this population yielding a new population. Finally, individuals of the original population are substituted by the new created individuals. This replacement usually tries to keep the best individuals deleting the worst ones. The whole process is repeated until a certain termination criterion is achieved (usually after a given number of iterations).

Notice that the algorithm establishes a trade-off between the exploitation of good solutions (selection stage) and the exploration of new zones of the search space (reproduction stage), based on the fact that the replacement policy allows the acceptation of new solutions that not necessarily improve the existing ones.

EAs are heuristics and thus they do not ensure an optimal solution. The behaviour of these algorithms is stochastic so they may potentially present different solutions in different runs of the same algorithm. That's why it is very common to need averaged results when studying some problem and why probabilities of success (or failure), percentages of search extension, etc... are normally used for describing their properties and work.

**I.2 Important Paradigms in Evolutionary Computation**

As it was shown in [Figure 1](http://www.lcc.uma.es/~ccottap/semEC/cap01/cap_1.html#Fi1), evolutionary algorithms have been historically divided into three categories. Their main differences come from the operators they use and in general from the way they implement the three mentioned stages: selection, reproduction and replacement.

|  |  |
| --- | --- |
| **Paradigm** | **Created by** |
| *Genetic Algorithms* | J.H. Holland |
| *Evolutionstrategie* | I. Rechenberg and H.P. Schwefel |
| *Evolutionary Programming* | L.J. Fogel, A.J. Owens, M.J. Walsh |

**Table 1.** *Paradigms in Evolutionary Computation.*

**I.3 Characterization of Evolutionary Computation**

The different nature-inspired algorithms can be formally described by a tuple of values. Thus we present the definition of the most usually found algorithms in evolutionary computation.

**GA(** *, G, , f, µ, I, S, Gap, , R, ***)**

|  |  |
| --- | --- |
| http://www.lcc.uma.es/~ccottap/symbols/sigma_uc.gif | the search space (phenotypes) |
| *G* | the representation space (genotypes, including alphabet and string length) |
| http://www.lcc.uma.es/~ccottap/symbols/rho_lc.gif: http://www.lcc.uma.es/~ccottap/symbols/sigma_uc.gifhttp://www.lcc.uma.es/~ccottap/symbols/arrow_r.gif*G* | is the representation (called *expression*) function |
| *f* : http://www.lcc.uma.es/~ccottap/symbols/sigma_uc.gifhttp://www.lcc.uma.es/~ccottap/symbols/arrow_r.gifR+ | is the fitness function |
| *µ* | size of the population |
| *I*: *µhttp://www.lcc.uma.es/~ccottap/symbols/arrow_r.gif*P(*G*) | initialization function |
| *S* | selection mechanism (roulette wheel, ranking, tournament, . . . ) |
| *Gap* | percentage of the population genetically operated on and inserted back |
| http://www.lcc.uma.es/~ccottap/symbols/omega_uc.gif | set of pairs (http://www.lcc.uma.es/~ccottap/symbols/omega_lc.gif, *Khttp://www.lcc.uma.es/~ccottap/symbols/omega_lc.gif*), i.e., genetic operators -*including crossover and mutation*- with their parameters (probability of application, . . . ) |
| *R* | replacement policy (worst, random, . . . ) |
| http://www.lcc.uma.es/~ccottap/symbols/tau_lc.gif | termination criterion (predefined number of steps, stagnation, . . .) |

**ES(** *µ, , , R, f, X, ,, ***)**

|  |  |
| --- | --- |
| *µ* | size of the population |
| http://www.lcc.uma.es/~ccottap/symbols/lambdalc.gif | number of offspring created in every step |
| http://www.lcc.uma.es/~ccottap/symbols/ell.gif | length of every string, i.e., number of (*parameter*, http://www.lcc.uma.es/~ccottap/symbols/sigma_lc.gif, http://www.lcc.uma.es/~ccottap/symbols/theta_lc.gif) triplets in every individual |
| *R* | replacement policy (plus, comma) |
| *f*: R^http://www.lcc.uma.es/~ccottap/symbols/ell.gifhttp://www.lcc.uma.es/~ccottap/symbols/arrow_r.gifR+ | is the fitness function |
| *X* | a recombination operator (none, local intermediate, . . . ) |
| http://www.lcc.uma.es/~ccottap/symbols/delta_uc.gifhttp://www.lcc.uma.es/~ccottap/symbols/sigma_lc.gif | step-size for modifying each individual's standard deviation vector http://www.lcc.uma.es/~ccottap/symbols/sigma_lc.gif |
| http://www.lcc.uma.es/~ccottap/symbols/delta_uc.gifhttp://www.lcc.uma.es/~ccottap/symbols/theta_lc.gif | step-size for modifying each individual's correlated mutation control parameter |
| http://www.lcc.uma.es/~ccottap/symbols/tau_lc.gif | termination condition (optimum is found, predefined number of steps, ...) |

**EP(** *, G, , f, µ, I, S, Gap, , R, ***)**

|  |  |
| --- | --- |
| http://www.lcc.uma.es/~ccottap/symbols/sigma_uc.gif | the search space (phenotypes) |
| *G* | the representation space (genotypes, including alphabet and string length) |
| http://www.lcc.uma.es/~ccottap/symbols/rho_lc.gif: http://www.lcc.uma.es/~ccottap/symbols/sigma_uc.gifhttp://www.lcc.uma.es/~ccottap/symbols/arrow_r.gif*G* | is the representation (called *expression*) function |
| *f* : http://www.lcc.uma.es/~ccottap/symbols/sigma_uc.gifhttp://www.lcc.uma.es/~ccottap/symbols/arrow_r.gifR+ | is the fitness function |
| *µ* | size of the population |
| *I*: *µhttp://www.lcc.uma.es/~ccottap/symbols/arrow_r.gif*P(*G*) | initialization function |
| *S* | selection mechanism (roulette wheel, ranking, tournament, . . . ) |
| *Gap* | percentage of the population genetically operated on and inserted back |
| http://www.lcc.uma.es/~ccottap/symbols/omega_uc.gif | set of pairs (http://www.lcc.uma.es/~ccottap/symbols/omega_lc.gif, *Khttp://www.lcc.uma.es/~ccottap/symbols/omega_lc.gif*), i.e., a set of mutation operators with their parameters (probability of application, . . . ) |
| *R* | replacement policy (worst, random, . . . ) |
| http://www.lcc.uma.es/~ccottap/symbols/tau_lc.gif | termination criterion (predefined number of steps, stagnation, . . .) |

Evolutionary Programming is very related to Evolution Strategies. Unlike Genetic Algorithms, no crossover operator is used in EP. Moreover, more emphasis is placed on behavioral changes rather than in the modification of the genetic material. For this reason, the genotype is usually very sophisticated (e.g., a finite automaton), and mutation operators are prepared to deal with such representations. It is also common to use several different mutation operators within the same algorithm.

**I.4 Common Components in EC Algorithms**

Any evolutionary algorithm is composed of a set of common elements in spite of its differences with respect to the rest of EC algorithms. The most important among these common elements are the following ones:

A population of *N* strings to work with. Typically strings are composed of binary, float or some complex structure (e.g. a tree) genes.

A fitness function to be optimized (either maximization or minimization) for evaluation of a string. It is usual for the fitness function to be called "the environment". In its simplest form, it maps the contents of a string to a real value. The function may be as simple as a polynomial or so complex as a whole system: neural network, some kind of system simulator, etc...

Some selection mechanism in order to simulate the survival of the fittest strings. A fitness-proportionate selection is commonly used in EC. Other types of selection include random selection of a string, ranking attending to fitness and selection with respect to position in the rank, tournament (select *K* strings randomly and choose the best among them) and many other variants.

Some replacement policy in order to keep population size constant. This is normally achieved by replacing the worst individual in the population. However many different choices are available in current works on EC, namely replacing some randomly choosen individual (the best can be lost), or else choosing a random set of *K* strings and replacing with the new string the worst of this set of strings. Elitism (the best always survives) is usually preferred.

Nature-inspired operators for changing a string into a new string. Such an operator is very frequently a mutation operation (replace a gene by another random gene). Another operator typically found in genetic algorithms is the crossover operator that crosses slices of two strings in order to compute two new strings. Hybrid and problem dependent operators are nowadays growing in successful applications.

**I.5 A Distinguished EC Representative: Genetic Algorithms**

Among the evolutionary techniques, the genetic algorithms (GAs) are the most extended group of methods representing the application of evolutionary tools. They rely on the use of a selection, crossover and mutation operators. Replacement is usually by generations of new individuals (see chapter 2 for more details).

Intuitively a GA proceeds by creating successive generations of better and better individuals by applying very simple operations. The search is only guided by the fitness value associated to every individual in the population. This value is used to rank individuals depending on their relative suitability for the problem being solved. The problem is the fitness function that for every individual is encharged of assigning the fitness value.



**Figure 4**. *A GA Proceeds in an Iterative Manner.*

The figure below presents an example of using the selection and crossover operators on a simple function. It is called the *working sheet of the GA*. In the beginning of the GA history mutation was considered as a secondary operator but at present many problems are successfully addressing only when mutation has a prominent role in the search of the algorithm.



**Figure 5**. *Working Sheet of a GA.*

**I.6 Relationship with other Techniques**

The location of this kind of techniques with respect to other deterministic and non-deterministic procedures is shown in the following tree. This figure outlines the situation of natural techniques among other well-known search procedures.



**Figure 6**. *A possible Classification of Search Techniques.*

Combinations of EAs with Hill-Climbing algorithms are very powerful (e.g., see [(Mühlenbein, 1989)](http://www.lcc.uma.es/~ccottap/semEC/APPREFS/Apprefs.html#Müh89)). Genetic algorithms intensively using such local search mechanism are termed *Memetic Algorithms* [(Moscató, 1989)](http://www.lcc.uma.es/~ccottap/semEC/APPREFS/Apprefs.html#Mos89). Also parallel models increase the extension and quality of the search. The EAs exploration compares quite well against the rest of search techniques for a similar search effort. Exploitation is a more difficult goal in EAs but nowadays many solutions exist for EAs to refine solutions.

**I.7 Important and Present Research Areas in Evolutionary Computation**

|  |  |
| --- | --- |
| A | [Air-Injected Hydrocyclone Optimization](http://www.lcc.uma.es/~ccottap/semEC/cap03/cap_3.html#C317)Artificial IntelligenceAssignation of Radio-Link Frequencies[Automated Parameter Tuning for Sonar Information Processing](http://www.lcc.uma.es/~ccottap/semEC/cap03/cap_3.html#C3110) |
| B | Bin Packing |
| C | ClusteringCommunication Network Design[Conformational Analysis of DNA](http://www.lcc.uma.es/~ccottap/semEC/cap03/cap_3.html#C319) |
| D | Data Mining[Dynamic Anticipatory Routing in Circuit-Switched Telecommunications Networks](http://www.lcc.uma.es/~ccottap/semEC/cap03/cap_3.html#C312) |
| E | Electronic-Circuit Layout |
| F | Flow Control[Fuzzy Controller Design](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#FLC) |
| G | Gas-Pipeline Control[Genetic Synthesis of Neural Network Architecture](http://www.lcc.uma.es/~ccottap/semEC/cap03/cap_3.html#C316) |
| H | Hybrid EC Systems |
| I | Image Generation and Recognition[Interdigitation (Engineering Design Optimization)](http://www.lcc.uma.es/~ccottap/semEC/cap03/cap_3.html#C3111) |
| J | Job Shop Scheduling |
| K | Knowledge Acquisition |
| L | Learning |
| M | Mathematical and Numerical Optimization[Models of International Security](http://www.lcc.uma.es/~ccottap/semEC/cap03/cap_3.html#C314)[Multiple Fault Diagnosis](http://www.lcc.uma.es/~ccottap/semEC/cap03/cap_3.html#C318) |
| N | [Neural Network Design](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#ANN)[Nonlinear Dynamical Systems](http://www.lcc.uma.es/~ccottap/semEC/cap03/cap_3.html#C314) |
| O | [Ordering Problems (TSP, N-Queens, . . . )](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#Ordering) |
| P | Parallel Process Scheduling[Parametric Design of Aircraft](http://www.lcc.uma.es/~ccottap/semEC/cap03/cap_3.html#C311)Portfolio Optimization |
| Q | [Query Optimization in Databases](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#Query) |
| R | Real Time Control of Physical Systems[Robot Trajectory Generation](http://www.lcc.uma.es/~ccottap/semEC/cap03/cap_3.html#C313) |
| S | [Sequence Scheduling (Genetic Edge Recombination)](http://www.lcc.uma.es/~ccottap/semEC/cap03/cap_3.html#C3112)[Strategy Acquisition](http://www.lcc.uma.es/~ccottap/semEC/cap03/cap_3.html#C315)Symbolic Integration and Differentiation |
| T | Time-Serie Analysis and Prediction[Traveling Salesman (Genetic Edge Recombination)](http://www.lcc.uma.es/~ccottap/semEC/cap03/cap_3.html#C3112) |
| U |  |
| V | [Validation of Communication Protocols](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#Protocols)VLSI Design |
| W | WYSIWYG Artistic Design |
| X | X-Ray Crystallography |
| Y |  |
| Z |  |

**This page was last updated on 20-Apr-98**

|  |  |
| --- | --- |
| **II.** | **GENETIC ALGORITHMS** |

**Contents of this chapter**

**II.1** Basic Operations

**II.2** Schema Theorem and Theoretical Background

**II.3** Solving a Problem: Genotype and Fitness

**II.4** Models of Evolution

**II.5** Advanced Operators

**II.6** Non-Conventional Genotypes

**II.7** Important Issues in the Implementation of a GA

**II.8** Response to Selection

**II.1 Basic Operations**

The basic operations first performed by a GA are the generation and evaluation of an initial population. Subsequently, a main loop of operations that include selecting the best individuals, crossing them and applying mutation on every locus (string position) is done. The solution is usually the best string that is present in the final population.

The computational effort in a normal GA usually resides in the evaluation of the initial and new strings. Normally the fitness function is complex: a mathematical function with many parameters, a full neural network that needs to inspect dozens of patterns for computing one single (float) fitness value, the simulation of a given system that yields a figure of merit of the string for this system, etc... Operators like selection, mutation, crossover or replacement are of linear complexity and work at constant rates for a given problem.



**Figure 1**. *GA at Work.*

A traditional GA works on binary strings of fixed length and applies fitness proportionate selection, one point crossover and bit-flip mutation. The initial population is generated by following an uniform random distribution, that is, every position in every string contains one 0 or one 1 with the same probability (0.5). The evolution model is represented by successive generations of populations, i.e., a new population completely replaces the old one. The algorithm stops when a predefined number of generations have been reached. Finally the evaluation of any given string is made by mapping the genotype to the phenotype by means of a binary-to-decimal operation for every group of bits of the genotype representing one problem parameter.

**II.2 Schema Theorem and Theoretical Background**

The theoretical basis of genetic algorithms relies on the concept of *schema* (pl. *schemata*). Schemata are templates that partially specify a solution (more strictly, a solution in the genotype space). If genotypes are strings built using symbols from an alphabet *A*, schemata are strings whose symbols belong to *A* U {\*}. This extra-symbol must be interpreted as a wildcard, being loci occupied by it called *undefined*. A chromosome is said to match a schema if they agree in the defined positions. For example:

The string 10011010 matches the schemata 1\*\*\*\*\*\*\* and \*\*011\*\*\* among others, but does not match \*1\*11\*\*\* because they differ in the second gene (the first defined gene in the schema).

A schema can be viewed as a hyperplane in a -dimensional space representing a set of solutions with common properties. Obviously, the number of solutions that match a schema *H* depend on the number of defined positions in it (its order, represented o(*H*)). Another related concept is the *defining-length* (represented (*H*)) of a schema, defined as the distance between the first and the last defined positions in it. See Table 1 for examples.

|  |  |  |
| --- | --- | --- |
| **Schema *H*** | **o(*H*)** | http://www.lcc.uma.es/~ccottap/semEC/cap02/delta_lc.gif**(*H*)** |
| 1\*\*\*\*\*\*\* | 1 | 0 |
| \*\*011\*\*\* | 3 | 2 |
| \*1\*11\*\*\* | 3 | 3 |

**Table 1.** Examples of schemata.



**Figure 2**. *The Concept of Schema.*

The GA works by allocating strings to best schemata exponentially through successive generations, being the selection mechanism the main responsible for this behavior. On the other hand the crossover operator is encharged for exploring new combinations of the present schemata in order to get fittest individuals. Finally the purpose of the mutation operator is to introduce fresh genotypic material in the population (it is specially important as generations proceed).



**Figure 3**. *Derivation of The Schema Theorem.*

All this behavior is analytically described by the so-called *schema theorem*, where *P*'s are the probabilities of crossover and mutation and *m(H,t)* is the number of instances (strings) the schema *H* has in the generation number *t*.



**Figure 4**. *The Schema Theorem.*

**II.3 Solving a Problem: Genotype and Fitness**

In order to apply a GA to a given problem the first decisions one has to make are concerning the kind of genotype the problem needs. That means that you must decide how the parameters of your problem maps into a binary string (if a binary string is to be used at all). This implies that you must choice the number of bits per parameter, the range of the decoded binary-to-decimal parameter, if the strings have constant or changing lengths during the GA operations, etc...

In traditional applications the strings use a binary alphabet and their length is constant during all the evolution process. Also all the parameters decode to the same range of values and are allocated the same number of bits for the genes in the string.

In most recent applications this situation is changing in a very important way. More and more complex applications need dynamic length strings or else a different non-binary alphabet. Typically integer or float genes are using in many domains as neural networks training, function optimization with a large number of variables, reordering problems as the Traveling Salesperson Problem, etc...

The traditional concept of a schema advises the utilization of binary genes because the number of schemata sampled per string is maximized. But a new interpretation of the schema concept has arrived to explain why many people have found very useful using other non-binary genes. This new interpretation [(Antonisse, 1989)](http://www.lcc.uma.es/~ccottap/semEC/APPREFS/Apprefs.html#Ant89) considers benefitial the utilization of higher cardinality alphabets due to their higher expression power (as many traditional AI paradigms consider). It is based on the consideration of the wildcard symbol (\*) as an instantiation character for different sets of symbols and not the traditional wildcard symbol. To be precise, \* is reinterpreted as a family of symbols \**x*, where *x* can be any subset of symbols of the alphabet. For binary strings these two interpretations are the same but for higher cardinality alphabets they are not.

Regarding genotypes, many other developments have pushed forward the utilization of non-string representations such as trees of symbols (e.g. Genetic Programming [(Koza, 1992)](http://www.lcc.uma.es/~ccottap/semEC/APPREFS/Apprefs.html#KO92)) that are much more appropriate for a large number of applications. Also extending every allele of the string along with the position (locus) it occupies in the string is interesting because we can freely mix the genes of the same string in order to get together the building blocks automatically during evolution. This is very interesting when we do not know which are the building blocks of our problem and want the GA to discover them. By associating the locus to every gene value we can reorder the string in the correct way before evaluating the string. These mechanism are common to the so-called *messy* GAs [(Goldberg, Deb, Korb, 1991)](http://www.lcc.uma.es/~ccottap/semEC/APPREFS/Apprefs.html#GDK91).

Here we arrive to the problem of decoding (called *expression*) the genotype to yield the phenotype, that is, the string of problem parameters. This is necessary since we need to evaluate the string in order to assign it a fitness value. Thus we use the phenotype as the input to the fitness function and get back a fitness value that helps the algorithm to relatively rank the strings in the population pool.

The construction of an appropriate fitness function is very important for the correct work of the GA. This function represents the problem environment, that is, it decides who much well the string solves the problem. Many problems arise when the fitness function has to be constructed:

 The fitness function depends on whether we want to maximize or minimize some criterion.

 The environment can present noise in the evaluations (partial evaluation for example).

 The fitness function may change dynamically as the GA proceeds.

 The fitness function might be so complex that only approximations to fitness values can be computed.

 The fitness function should allocate very different values to strings in order to facilitate the work of the selection operator.

 The fitness function must consider the constraints of the problem. Normally if unfeasible solutions can appear the fitness function must allocate a small fitness value (if it is a maximization problem).

 The fitness function might incorporate some different sub-objectives. Such a multi-objective function presents non-conventional problems when used in the GA.

 The fitness function is a black box for the GA. You feed it with the phenotype and then you get the fitness value. Internally this may be achieved by a mathematical function, a complex computer simulator program or a human used that decides who good a string is (it is normal when strings encode images or music or something alike).

Regarding dynamic fitness functions a genotype of diploid genes can be used in which every value is determined by some relationship among two alleles encoded in the same string. The extension to *K*-ploid strings is straightforward. A **dominance** operator needs to be defined in order to get a single gene (parameter) value from the *K* alleles that determine it.

As the evolution proceeds the fitness values assigned by the fitness function to the strings are very similar since the strings we are evaluating are also very similar. There exist some different **scaling** operators that help in separating the fitness values in order to improve the work of the selection mechanism. The most common ones are [(Michalewicz, 1992)](http://www.lcc.uma.es/~ccottap/semEC/APPREFS/Apprefs.html#MI92):

* Linear Scaling: *f'=af+b*
* Sigma Truncation: *f'=max[0, f-(f\*-c)],* where *f\** (resp. ) is the mean fitness value (resp. standard deviation) in the population.
* Power Scaling: *f'=(f)^k*

Also when very different strings get the same fitness values an important problem arises in that crossing them yield very bad individuals (called *lethals*) unable of further improvements. **Sharing** methods [(Goldberg, 1989)](http://www.lcc.uma.es/~ccottap/semEC/APPREFS/Apprefs.html#GOL89) help to allow the concurrent existence of very different solutions in the same genetic population. Also **parallel models** help in overcoming this problem.

Sometimes is very interesting to rank the population according to the fitness values of the strings and then to apply a reproductive plan that uses the rank position of the strings instead of the fitness values. This is good for ill-designed fitness function or when very similar fitness values are expected. This mechanism is called **ranking** [(Whitley, 1989)](http://www.lcc.uma.es/~ccottap/semEC/APPREFS/Apprefs.html#WH89), and keeps a constant selection pressure on the population.

**II.4 Models of Evolution**

In traditional GAs the basic evolution step is a generation of individuals. That means that the atomic advance step of the GA is one generation. There exists another interesting alternative called **Steady-State GAs** in which the atomic step is the computation of one single new individual. In these algorithms after generating the initial population, the GA proceeds in steps that consist of selecting two parents, crossing them to get one (or two) single individual and mutating the resulting offspring. The new string is inserted back in the population and one of the pre-existing strings (a random string or the worst present string) leave the pool (normally if the new string is worst than the actual worst it is not inserted at all).

This one-at-a-time reproduction was initially introduced to overcome problems in the training of neural networks and was extended from there to the rest of present applications. It represents an interesting equilibrium between the exploration and exploitation edges of a GA, because the GA maintains a pool of strings (exploration, schemata, etc...) and at the same time it performs some kind of hill-climbing since the population is only pushed torward better solutions. Elitism and ranking are very common to steady-state GAs.

Finally there exists an intermediate model of evolution for a sequential GA in which a percentage of the full population is selected for the application of genetic operations. This percentage is called the **GAP** and it represents the generalization of any evolution model since it includes the two apposite approaches (generations versus one-at-a-time) and any other intermediate model.

Many other models of evolution exist in the field of parallel GAs depending on their granularity, homogeneity, etc... but we will discuss them in a following chapter.

**II.5 Advanced Operations**

Besides the traditional operations, a GA can incorporate new techniques at different levels. One of the most used innovations is to define new genetic operators improving in some way the behaviour of the general one-point crossover or bit-flip mutation for certain problems. Normally these new operators are the result of a theoretical study of GAs or else they are new versions of traditional operators that incorporate problem-dependent knowledge.



**Figure 5**. *Some Examples of New Crossover Techniques: Single-Point, Double-Point, Uniform and Arithmetic Crossover Operators.*

Also advanced genotypes can be used instead of the traditional binary encoding. Finally new schemes of evolution (steady-state, parallelism, ...) or foreign operators (hillclimbing operators, existing algorithms, ...) can be added to the GA skeleton, being applied with a given probability of success as the next generation is being created from the previous one.



**Figure 6**. *A Steady-State GA Architecture with Different Operators.*

**II.6 Non-Conventional Genotypes**

As mentioned before, genetic algorithms have traditionally used a pure binary genotype in which every parameter of the problem is encoded in binary by using a given number of bits. The number of bits for every gene (parameter) and the decimal range in which they decode are usually the same but nothing precludes the utilization of a different number of bits or range for every gene.

In order to avoid the so-called *hamming cliffs* the gray code is sometimes used when decoding a binary string. With this code any two adjacent decimal values are encoded with binary numbers that only differ in one bit. This helps the GA to make a gracefully approach to an optimum and normally reduces the complexity of the resulting search.

Nevertheless, the plethora of applications to which GAs have been faced have motivated the design of many different genotypes. This means that the string can be constructed over an integer or real alphabet, for example. Integer alphabets have interesting properties for reordering problems such as the Traveling Salesperson Problem and others. On the other hand floating-point genes are very useful for numeric optimization (for example for neural network training). These higher cardinality alphabets enhance the search by reducing the number of ill-constructed strings resulting from the application of crossover or mutation. Also, when a very large number of parameters has to be optimized, it is helpful to rely on shorter strings (also because populations and the number of steps are smaller).

If the genotype can be expressed as a string, we need to decide for any given application if the genome length is going to be kept constant or allowed to change. Also, it is an interesting decision to use diploid individuals in dynamic environments (i.e., environments in which the fitness function changes as the search proceeds). In a diploid individual two -homologous- chromosomes describe the same set of parameters. When decoding the string some kind of *dominance* operator need to be used in order to decide the value of the problem parameter (gene).

In many recent applications more sophisticated genotypes are appearing:

 One individual can be a tree of symbols (like in Genetic Programming applications - see [Figure 7](http://www.lcc.uma.es/~ccottap/semEC/cap02/cap_2.html#Fi7)).

 The individual is a combination of a string and a tree.

 Some parts of the string can evolve and some others don't in different moments of the search.

 The individual can be a matrix of symbols.

 The genotype could be a string of symbols that indicate operations to be performed. A simulator interprets the symbols and allocates fitness values depending on the result of the interpretation of the string.



**Figure 7**. *Symbols Tree.*

**II.7 Important Issues in the Implementation of a GA**

It follows an initial summary of issues relating the implementation of a sequential GA:

 **Do not implement the population as an array of fixed length:** your implementation will not be flexible and sorting or insertions/replacements will bee slow operations. Instead use a list of pointers (specially if you are using a steady-state GA).

 **Do not make re-evaluations of individuals when the fitness value is needed.** This error is very common when people develops general GA libraries. This might be negligible for toy applications but real applications do not admit two or more evaluations of the same individual (because of very high computational costs).

 **Do not implement in traditional imperative languages a library for evolutionary algorithms.** Reuse, errors detection, comparisons and many other operations are readily easy if you use an object oriented language (and also an O. O. system design) for the implementation. ***Please notice that an implementation with an O. O. language is quite different from an O. O. GA system!.***

 **Do not implement with a Logic Language,** the implementation requires a lot of memory recurses and time of computation.

**II.8 Response to Selection**

In this section we summarize of the theory presented in [Mühlenbein and Schlierkamp-Voosen (1995)](http://www.lcc.uma.es/~ccottap/semEC/APPREFS/Apprefs.html#MS95). Let be the average fitness of the population at generation *t*. The response to selection is defined as
 .
The amount of selection is measured by the selection differential
 ,
where is the average fitness of the selected parents. The equation for the response to selection relates *R* and *S*:
 .
The value *b*(*t*) is called the *realized heritability*. For many fitness functions and selection schemes, the selection differential can be expressed as a function of the standard deviation of the fitness of the population. For *truncation selection* (selecting the *T·N* best individuals) and for normally distributed fitness, the selection differential is proportional to the standard deviation:
 .
The vaule *I* is called the *selection intensisty*. For arbitrary distributions one can show the following estimate:
 .
For normally distributed fitness the famous equation for the response to selection is obtained:
 .
The above equation is valid for a large range of distributions, not just for a normal distribution. The response depends on the selection intensity, th realized heritability, and the standard deviation of the fitness distribution. In order to use the above equation for prediction, one has to estimate *b*(*t*). The equation also gives a design goal for genetic operators -- to *maximize the product of heritability and standard deviation*. In other words, if two recombination operators have the same heritability, the operator creating an offspring population with larger standard deviation is to be preferred.
The equation defines also a design goal for a selection method -- to *maximize the product of selection intensity and standard deviation*. In simpler terms, if two selection methods have the same selection intensity, the method giving the higher standard deviation of the selected parents is to be preferred. For proportionate selection as used by the simple genetic algorithm [(Goldberg, 1989)](http://www.lcc.uma.es/~ccottap/semEC/Apprefs/Apprefs.html#GOL89) it was shown by Mühlenbein and Schlierkamp-Voosen (1993) that
 .
The equation shows that the selection intensity of proportionate selection goes to zero for *t* inf, whereas truncation selection has a constant selection intensity. Proportionate selection selects too weak in the final stages of the search.

**This page was last updated on 06-Apr-98**

|  |  |
| --- | --- |
| **III.** | **APPLICATIONS OF GENETIC ALGORITHMS** |

**Contents of this chapter**

**III.1** Index of Most Important Applications of the Genetic Algorithms

**III.2** GAs for the Design of Neural Networks

**III.2** GAs in Combinatorial Optimization

**III.1 Index of Most Important Applications of the Genetic Algorithms**

This section presents a overview of applications of genetic algorithms to real-world problems.

**Genetic Algorithms in Parametric Design of Aircraft**, by Mark F. Bramlette and Eugene E. Bouchard. The authors discuss optimizing aircraft designs when the task is posed as that of optimizing a list of parameters. They have approached the problem with a number of optimization algorithms, including a genetic algorithm using real number representation. They also discuss the performance of each algorithm and describe some innovative techniques used in their quite successful genetic algorithm, including the technique of generating a large number of initial population members and then working only with the best ones.

**Dynamic Anticipatory Routing in Circuit-Switched Telecommunications Networks**, by Louis Anthony Cox, Jr., Lawrence Davis, and Yuping Qiu. The objective of the study is to optimize the routing of telephone networks in order to minimize costs to US West. It compares the performance of an order-based genetic algorithm with several other optimization techniques on this problem. The authors conclude that the genetic algorithm is a highly successful technique when the problem is complex, but hybridization of these algoritms can lead to better performance than using any of them in isolation.

**A Genetic Algorithm Applied to Robot Trajectory Generation**, by Yuval Davidor. He shows how to apply genetic algorithm techniques to the task of planning the path which a robot arm is to take in moving from one point to another. Davidor uses variable-length chromosomes in his solution, and devises some novel and interesting crossover operators.

**Genetic Algorithms, Nonlinear Dynamical Systems, and Models of International Security**, by Stephanie Forrest and Gottfried Mayer-Kress, concerns a problem posed by current research in chaotic models of real processes. Chaotic models of international arms races and economic competition seem to model some features of the real-world processes better than some other more traditional models have done. The authors use a genetic algorithm to find good settings of the parameters of Mayer-Kress's models in order to enhance their performance on the models.

**Strategy Acquisition with Genetic Algorithms**, by John J. Grefenstette. He experiments with SAMUEL, a genetic algorithm that learns techniques for maneuvering a simulated airplane in order to evade a simulated missile. The genetic algorithm he describes employs several techniques of interest, including variable-length chromosomes composed of rules that form a production system. A chromosome is evaluated by using those rules to maneuver the airplane in simulated interactions between airplanes and missiles. Grefenstette has built knowledge of the production rule domain into his operators in clever ways.

**Genetic Synthesis of Neural Network Architecture**, by Steven A. Harp and Tariq Samad, that describes techniques for encoding neural network architectures on binary chrmosomes. The authors use variable-length chromosomes and a variety of other novel techniques. This is a good place to begin in learning how to combine neural networks and genetic algorithms.

**Air-Injected Hydrocyclone Optimization Via Genetic Algorithm**, by Charles L. Karr, that describes the solution of a design problem by a genetic algorithm using the bit string representation technique. Karr represents the design of an air-injected hydrocyclone as a list of parameters. An interesting feature of his approach is the use of a new operator called "simplex reproduction". Karr shows that a genetic algorithm using this operator is quite effective as a search technique for finding design parameter combinations.

**A Genetic Algorithm Approach to Multiple Fault Diagnosis**, by Gunar E. Liepens and W. D. Potter, that discusses the use of a genetic algorithm for finding the most plausible combination of causes for alarms in a microwave communication system. The authors use binary chromosomes to represent solutions to a problem that they show is a type of set covering problem. They show how to incorporate knowledge about set covering optimization into their genetic algorithm in novel ways, yielding a high-performance hybrid solution to the problem.

**A Genetic Algorithm for Conformational Analysis of DNA**, by C. B. Lucasius, M. J. J. Blommers, L. M. C. Buydens, and G. Kateman. It is a development of a genetic algorithm for determining the structure of a sample of DNA based on spectrometric data about the sample. An interesting "cascaded" evaluation technique that greatly enhances the efficiency of their evaluation function is used. The authors use bit strings to encode molecular structures. Their evaluation function measures the degree to which each decoded structure conforms to the data that have been collected about the sample. The genetic algorithm evolves a description of molecular structure that is in agreement with the data collected. The problem of determining biomolecular structure occupies a central position in the worlds of fundamental and applied chemistry today.

**Automated Parameter Tuning for Sonar Information Processing**, by David J. Montana. An application of genetic algorithms to two problems associated with interpreting passive sonar data. The first is a parameterization problem. To solve it, Montana uses a floating-point version of OOGA to find good parameter settings for the algorithms employed in the process of interpreting sonar data. The second problem is a classification problem. For this problem, a genetic algorithm is used to train neural networks classifying sonar signals in various ways. In this second system, Montana and Davis experiment with a number of domain-based operators, including the use of backpropagation -a neural network technique- as a genetic algorithm operator. This appliction is useful if you are interested in hybrid genetic algorithms, real number representations for parameterization, or neural networks.

**Interdigitation: A Hybrid Technique for Engineering Design Optimization**, by Gilbert Syswerda. An application of a genetic algorithm to the problem of scheduling activities in a laboratory in which each activity may affect the others in a variety of ways. Syswerda has been implementing this system under contract to the U. S. Navy. The genetic algorithm uses an order-based chromosome to represent its schedule. The chromosome is decoded with a decoder that incorporates a good deal of knowledge about the scheduling domain.

**The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination**, by Darrell Whitley, Timothy Starkweather, and Daniel Shaner. The authors describe a technique for solving the traveling salesman problem, a well-known combinatorial optimization problem. Their solution includes novel and ingenious representation, crossover, and repair mechanisms. They also show how similar techniques can be applied to the problem of scheduling a Hewlett-Packard production line.

**III.2 GAs for the Design of Neural Networks**

[(Alba, Aldana, Troya, 1993)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#AL93)

**III.2.1 *Introduction***

Artificial Neural Networks (ANNs) represent an important paradigm in AI dealing with massively parallel information processing proposed as biological models for the human brain. ANNs are widely used to offer human-like skills where they are needed, so we can find them in pattern recognition, signal processing, intelligent control and many other applications that can be faced by introducing a network as the heart of the solution system (see [(Kim, Lee, Kim, Hwang, 1991)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#KI91)).

Wherever an ANN is to be used it must first be designed. At present, any ANN design drags along an unstructured, heuristic and arbitrary path in order to reach "the better" structure and connectivity to be trained. Only the training methods are truly applied, but every ANN type seems to need a different own training mechanism. Usually the training mechanism is some kind of hillclimbing prosecution, that is very closely related to (and so, dependent on) the problem being solved, the ANN type and/or the pattern set for it. The final result is a vast landscape of different multiparameter tuning procedures to be carried out for any individual problem and with no warranties for optimum results.

This lack of methodology and the hope of a general multifaced quasioptimum training made Genetic Algorithms (GAs) suitable for our purposes. Defined by J. Holland in 1975 GAs simulate natural evolution. In [(Goldberg, 1989)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#GOL89), [(Whitley, 1989)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#WH89b) and [(Whitley, Starkweather, Bogart, 1990)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#WH90) can be found the basis of our GA and ANN approaches. For our purposes chromosomes will encode ANNs, genes will encode the items being optimized (weights, links or hidden layers) and alleles will be gene components (regarding upon the used coding one or more alleles are included to compose a single gene value). Initial strings will be genetically evolved over generations of newly created offsprings searching for an optimum.

Because any ANN must be coded as a chromosome string ANN independence is achieved, just some *evaluation* procedure must be defined to recognize relative fitness of individuals to the problem. GA techniques have a stochastic behaviour and so we only can expect **quasioptimum** (very frequent good or optimum) trainings. Besides local minima avoiding, generality, multiple points parallel search and robutness we can get further in using GAs to complete ANN design. Since GAs work on some coding of the solution and not on the solution itself, we can code "any" problem as a string of parameters and submit it to genetic optimization. Thus, it is only necessary to properly code ANN *connectivity and structure* as strings and define an evaluation procedure to get two new levels of ANN design. This way we can bring optimization methodology to these two design stages. This **full three level design** is thought to help designer's work from the problem specification (patterns) up to a quasioptimum ANN to solve it (that is called a **Genetic ANN**). These three levels of design will be fully accomplished by using Genetic Algorithms. An introduction to genetic ANNs can be found in [(Mühlenbein 1989)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#MU89) and in [(Mühlenbein 1990)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#MU90).



**Figure 1.** *Three Level GANN Design.*

We have designed and built up a genetic tool called **G.R.I.A.L.** (**G**enetic **R**esearch **I**n **A**rtificial **L**earning) [(Alba, Aldana, Troya, 1992)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#AL92) to implement several known and new GA techniques and the three level genetic ANN design in order to test their efficacy and properties. We are furthermore concerned with scalability and computational efficiency, thus we have used a Concurrent Logic Language called **PARLOG** to implement GA and ANN behaviour in GRIAL as a new kind of computational approach in the aim of profiting from the outcoming parallelism advantages.

The used test suite is made up of four problems. The XOR and the Two-bits Binary Adder (TBA) problem (but with 4 patterns) as presented in [(Whitley, Starkweather, Bogart, 1990)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#WH90). A feedforward network to make Spanish Plurals (three classes classification) and a Hopfield network to behave as a 9-characters recognitor. The XOR and TBA problems are fully tested (training, connectivity and layered structure, either separately and together). The Spanish Plurals ANN has been trained and structure-optimized and the Hopfield network has been trained. While trying to get the optimum ANN for every problem we have tested the relative influence of the multiple GRIAL techniques.

In this work we have tested in GRIAL the effects of the *traditional selection* procedure versus a *one-at-a-time selection* procedure. We explore the influence of coding on ANN design by using *binary, real and a diploid genotypes* (this last never tested before for ANN design). A *migration* scheme and an *adaptive mutation* similar to those used in [(Whitley, Starkweather, Bogart, 1990)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#WH90) are tested against sequential single GAs and constant mutation. A smooth bit climber-like operator [(Davis, 1991)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#DA91) called *GBit* is tried and a *Mating Restriction* similar to [(Deb, Goldberg, 1989)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#DE89) is implemented. Dislike partial genetically defined ANNs (as [(Whitley, Starkweather, Bogart, 1990)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#WH90) where genetic training and connectivity are addressed as separate works and [(Harp, Samad, Guha, 1989)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#HA89) where structure is genetically defined but backpropagation is used for training) we have designed a full genetic and automatic ANN designer in order to optimize any ANN component.

**III.2.2 *PARLOG***

PARLOG [(Clark, Gregory, 1986)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#CL86) is a Concurrent Logic Language which has been developed at the Imperial College. Operationally the computational model of this kind of language consists in a concurrent processes set which communicates by means of binding logic variables and which synchronizes by waiting for unbounded logic variables. The possible behaviors of a process are defined by means of guarded horn clauses: **Head <- Guard : Body.** Head and Guard defines the conditions under which a reduction can be made. Body specifies the resulting state of the processes after the reduction.

Parlog is one of these languages which exploit two kinds of parallelism: stream and-parallelism and or-parallelism. The first type of parallelism occurs when a goal is reduced to a conjuction of subgoals and they are all tested in parallel. The second type of parallelism appears when a predicate can be solved by more than one clause. In this case, all of them are tested at the same time and, if more than one fulfils its guard, one of them will be selected in an indeterministic way. Parlog also has some primitives which can be used to avoid both types of parallelism (see [(Crammond, Davison, Burt, Huntbach, Lam, 1989)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#CR89)).

This kind of language fits very well in the parallel programming paradigm. In opposition to sequential logic languages which present a transformational behaviour, concurrent logic languages are well fitted to the specification of reactive system, that is, of open systems which have a high level of interaction with their environment. And, as stated in [(Troya, Aldana, 1991)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#TR91) , this is what a neural network does: it tries to reach a statistical minimum that is environment- dependent.

GRIAL is oriented to provide an easy changing of the GA strategies used to solve a given problem. *Unix parallelism* is achieved at *interlevel* communications while *Parlog parallelism* appears at *intralevel* searchs. Real LAN distribution of intralevel Parlog parallelism can be faced by using Parlog mail-boxes. Parlog allows entering parallelism from string managing genetic operations till ANN neurons activations. Fine and coarse grained approaches are of a straightforward implementation with Parlog. These advantages along with its lists and symbols processing make Parlog a better language than imperative ones to get good and reliable implementations.

**III.2.3 *Full Genetic ANN Design***

In this section a three level and full genetic ANN design is presented and analyzed by means of GRIAL. New and existing GA techniques have been tested to get a qualitative understanding of the properties of this kind of design. We envisage the following exposition from bottom (genetic training) to up (genetic structure definition) passing through an intermediate genetic connectivity definition level.

**III.2.3.1 Genetic Training**

To submit any ANN type to genetic training we must define some proper coding for weights to appear as strings. GA's work needs some local logic meaning to be present in strings, i.e. a chromosome must be coded to include *logic building blocks* with some meaning for the underlying problem [(Goldberg, 1989)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#GOL89). Then we code any ANN string as being a sequence of its input link weights to every neuron in the ANN, from the input layer to the output layer. The genetic crossover of two different strings (coded ANNs) profits from their best slices to create better trained offsprings. Through natural selection bad *schemata* (bad structures of the solution) are exponentially discarded and good schemata are exponentially reproduced as evolution occurs. To evaluate the fitness of a string to the *environment* (problem) strings are *expressed* (decoded) as ANNs. We use *SQE* (squared error between desired and obtained outputs extended to any output neuron and any pattern in the pattern set) as the fitness measurement to help natural selection's work: stochastic selection picks up the best strings to be crossed (reproduced) to yield offsprings in next generation.

Weights can be coded in strings attending to several codes. **Binary** code (signed magnitude) is very extended in genetics. **Real**, **Reordering** and **Diploid** [(Goldberg, 1989)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#GOL89) codings are another known ones we have tried for ANN training. Binary code is very good for GA job but for ANN training it needs too large populations and evolutions even for very small problems (we want to keep population size on hundreds of individuals). Reordering schemes (genes, PMX and/or Inversion) do not seem to improve binary results, and we think this is because we are using a correct genotype representation of the problem that does not need additional genetic help. Real codings (one-weight/one-real-gene) seems to be the true useful genotype because they allow small and quick GAs to solve adequately the trainings, despite they present an undesired low diversity maintenance during evolution that provoques local minima appearences. All these codings are *Haploid* codings (one string encodes one ANN), but *Diploid* chromosomes with triallelic values [(Goldberg, 1989)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#GOL89) (and maybe with real values...) have much to say in allowing sophisticated behaviour by helping diversity and natural adaptation to traumatic changes in the environmental conditions (they outperform the other codings using half the number of strings).

Any feedforward, recurrent, dynamic, static or any other ANN type can be trained by GA means. We have tried *constant* and *adaptive mutation* operators (this last based on Hamming Distance -hd-) to maintain diversity in population. We think that mutation is not a secondary operator but an essential technique, very useful in keeping population size at a relatively low value. Probability for adaptive mutation is computed as 1/hd (linear) and we have detected this as a somewhat high value (high mutation has allowed good searches with our one-at-a-time selection), but in GRIAL, a control technique called *begin-end-frequency* allows a better pursuit of GA strategies' effects by specifying how and when to apply them. In order to speed up the search of a quasioptimum weights set we have designed the **GBit** operator, a hybrid genetic-hillclimbing procedure that makes smooth changes in the best-to-now solution string to explore its neighborhood. Crossing two strings is not a trivial operation because these two strings may represent different functionality-neurons distributions for the problem solution and crossover will often yield two new strings (ANNs) that behave much worst than their parents and that are unable of future improvement (called *Lethal* strings). To solve this problem we have designed a **Restrictive Mating** operator based on hamming distance (similar to the Genotypic Mating Restriction described in [(Deb, Goldberg, 1989)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#DE89) to impose a minimum likeness between mates to be sure that interspecies (very different couples) crossovers do not take place. This problem is very frequent in big populations where many good but different partial solutions often appear. GBit and RMating have shown to turn GAs quick tools of high efficacy. Since ANN coding/decoding and evaluation operations are very complex and expensive we look for improved selection mechanisms that minimize wanderings along the search space while maintaining GA properties. The **Generations Evolutive Scheme** (a full new generation replaces the old one) using the *Stochastic Remainder Without Replacement* seems to be very expensive in our tests despite its advantages in preserving diversity and genotype exploration. That's why we have designed the **Immediate Effects Evolutive Scheme** to keep strings ranked (from best to worst) and using the *Roulette Wheel* selection to pick up two individuals to be genetically processed and produce offsprings to be inserted in the ranking. This later selection operator is the best choice for ANN design, but population size must be kept large enough to avoid premature convergence due to its more minimum-directed search. A complete-ranked selection as that in [(Whitley, 1989)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#WH89) could lessen genetic drift because in GRIAL we allow duplicates in the population and the IEES produces a high selection pressure.

We have finally tested and stated the superiority of any **Distributed search** using N GAs over any equivalent single GA search for the whole test suite. We have used a **Migration** scheme similar to [(Whitley, Starkweather, Bogart, 1990)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#WH90) to define N GAs in parallel evolution with occasional exchanges of their respective best strings through Parlog streams. This parallelism improves time, accuracy, diversity, lethals avoiding and is more likely to yield a quasioptimum result. Real speed-up for quick GA evolutions can be brought from real machine distribution of the N GAs in a LAN or multiprocessor computer.

**III.2.3.2 Connectivity Optimization**

In traditional ANN design connectivity is determined by the ANN type to be used or by experience. We want to make a *genetic connectivity definition* to accomplish three goals: *(a)* GAs are used to define connectivity and weights (training), *(b)* we want our method to be scalable with problem size and *(c)* the predefined structure to be connected must be full used, because we expect this structure to have been selected as quasioptimum by any means.

By coding links in strings we want to obtain the GAs advantages not only in that links pruning is tried but links appear and disappear in parallel in strings as evolution occurs. Binary codings as the used in [(Whitley, Starkweather, Bogart, 1990)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#WH90) fill every string position to encode presence (1) or absence (0) of a link, but this coding yields excessively long chromosomes that do not scale as the problem grows because, even if the GA discovers the useless links, the strings *must hold room enough* to contain the full link space. To allow designer deciding the magnitude of the search to be carried out we only impose a low and high limit to GA strings length to encode links (regarding the ANN structure).

We have designed a **General Artificial Neuralnetwork** sheet (GAN) as an implicit link space for feedforward and/or recurrent ANNs. For a given ANN structure three GAN input link spaces are defined: **FF** (feedforward), **FK** (feedback) and **DIO** (direct-io) input link spaces. We associate every link with its weight value through the use of symbolic genes we call **wlink genes** to allow *independent existence* of any link in the ANN. Every string of wlink genes in the GA population encodes a *subset* of the full link space being optimized.

The user of GRIAL can specify the way and strings length in which link subsets are encoded. Thus, designer can specify the length of strings in the population, then *pruning* as desired the full link space. Excessive initial pruning has shown to be undesirable because networks are unable to learn the full pattern set. The rational use of the ANN structure brings from the results one can get by pruning links beyond a low limit: many neurons can be present in the final ANN whose input and/or outputs are not being considered when the ANN works. So we have defined **Archetypes** as distinguished string positions respected by the crossover during evolution. Archetypes assure that the ANN structure is being profited by

This linking (L) level uses the training (W) level to determine weights and fitness for its initial population and evolution is responsible for crossing the subsets of links (strings) looking for a quasioptimum solution. Link pruning and adding are achieved by a *link duplication effect*. We have tried penalty schemes attending to links number in order to modify the fitness values, but results indicate that any other penalty schemes (as giving more -W- learning facilities to strings) should be of greater success. But the real power of this technique becomes from the initial designer-drived pruning and the subsequent GA optimization. Again the best results are these obtained with N distributed GAs working in parallel. At L level we pretend to make a more natural ANN design by interacting with the training lower level and then making it easy as well as more accurate and cheaper to implement (as software or hardware ANNs).

**III.2.3.3 Structure Definition**

Defining the best ANN structure for a given problem is not a deterministic nor even a well known process due to the complexity of recognizing every neuron job in the whole ANN and the relationships among neurons. There exist many rules based on experience about the number and disposition of neurons, but, as for connectivity definition, designers have not a methodology out of the proof-and-error mechanics.

The real problem for a genetic structure definition is *to select a good coding* to contain structure information, general enough to be combined by crossover and specific enough to determine one unique ANN structure when decoded (expressed). We have designed a **pure binary genotype** to do this job (a *strong* coding of the structure). We want to optimize the number of hidden layers and neurons per hidden layer. GRIAL puts in a chromosome one gene for every one of the **GN** potentially existing hidden layers, and the genes length (**GL**) determines the maximum number of neurons per hidden layer. Then if we want between 0 and GN hidden layers and between 0 and 2GL-1 neurons per hidden layer we should use binary chromosomes of GN\*GL length in the *Structuring* (S) level.

We have got very good results for feedforward and GAN networks regarding useless hidden neurons pruning (in order to reduce the cost per ANN) and quasioptimum structures generation (e.g. we have genetically duplicated the human proposed structure for the XOR problem). Furthermore a **penalty scheme** based on changes in the fitness values according to the neurons number of the network has been successfully tried with some interesting side effects in helping natural selection decisions while searching for smaller and cheaper ANN structures.

Usually the kind of desired structure is best known by the designer than the best connectivity pattern or even the weights set to be used, and that's why S level search is accomplished by smaller GAs than those GAs needed at L or W levels. On the other hand S strings are very complex and expensive to evaluate for fitness because they require multiple L and/or W GAs executions. Genetic operators have been extended to this level (e.g. GBit or mutation) but the simple traditional GA has shown to be a good procedure for S level. We encourage the use of the *Immediate Effects Evolutive Scheme* in order to prevent useless wanderings even when we risk to obtain only good structures and not the best ones. For structure definition, hillclimbing-like methods may reduce the overall cost.

The final best ANN will have been designed taking account for many aspects to yield a full suited ANN to the proposed environment (the problem). Individual GAs will gain by being allowed to present a higher rate of error due to the multiple level relationships (GA parameters tuning does not need to be of high accuracy to work well). As we can see the parallelism at many computational levels is the best way to bring efficiency and efficacy to this complex Genetic ANN design: three levels run in parallel, several GAs can work in parallel at any level and every genetic operation can be parallelized due to PARLOG software enhancement. PARLOG can be used to define an Object Oriented library of ANN simulators [(Troya, Aldana, 1991)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#TR91) and to implement any kind of GA parallelism as those mechanisms outlined in [(Macfarlane, East)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#MAES). GRIAL includes some of these propositions.

**III.2.4 *Conclusions***

There exist many different non GA ways to define the target ANN to solve a problem and many other Evolutive mechanisms but the **Full Three Levels ANN Design** is thought to be the best and more natural way to take designer to quasioptimum ANNs by controlling only some GA strategies. GAs provide the better set of tools to do this work. Resulting ANNs can be later refined by any of the appliable existing mechanisms.

We think that a smart connectivity for initial pruning to be used at L strings (initial link strings composition) can improve the connectivity definition, because, when full automatic design is being performed, S level randomly generates this pruning and only a high number of strings at S level can overcome excessive initial pruning (because numerous initial prunings are to be tested).

The "best" GA to ANN design is a distributed search with IEES or a similar ranked selection, using a somewhat high mutation rate, two point crossover and some help to avoid lethals and to speed up search (GBit or another bit climber and some niche & species formation technique like RMating if no migration is to be used).

ANNs and GAs are thought to be open and reactive systems which fit very well whithin the logic and the concurrent programming paradigms. Thus, many advantages regarding parallel implementations and software improvements can be brought from the use of Parlog as the base language.

With the correct combination of our GA techniques or some extensions of them we can define a general methodology to enter automatic ANN design in a unified fashion while still maintaining diversity of approaches to solve a problem. Computational requirements of this kind of design (through genetics) are to be enhanced by parallelism in order to reduce the time and memory consuming GAs that we are still forced to use in solving large problems.

**III.3 GAs in Combinatorial Optimization**

There exists an extensive range of problems which can be formulated as obtaining the values for a vector of variables subject to some restrictions. The elements of this vector are denominated decision-variables, and their nature determines a classification of this kind of problems. Specifically, if decision-variables are required to be discrete (i.e. the solution is a set or sequence of integer numbers or any other discrete type), the problem is said to be combinatorial. The process of finding optimal solutions (maximizing or minimizing an objective function) for such a problem is called combinatorial optimization.

Combinatorial optimization problems have been traditionally approached using exact techniques such as Branch and Bound [(Lawler and Wood, 1966)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#LW66). Finding the optimal solution is ensured with these techniques but, unfortunately, they are seriously limited in their application due to the so-called *combinatorial explosion*. As an example, consider the Travelling Salesman Problem (TSP). This problem (obtaining a minimal Hamiltonian tour through a complete graph of n nodes) is a classical example of NP-complexity: the work-area to be explored grows exponentially according with the number of nodes in the graph, and so does the complexity of every know algorithm to solve this problem. It is not only a good example of a combinatorial optimization problem, but also an approach to real problems like VLSI-design or X-ray crystallography [(Reinelt, 1994)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#RE94).

**III.3.1 *Genetic Algorithms to solve the TSP***

Building a genetic algorithm to solve the TSP requires specifying those elements described in the GA definition. However, here only the really TSP-specific elements will be described since others (such as selection and replacement functions) are normally problem-independent.

**III.3.1.1 Fitness function**

This element is here very simple. In fact, it suffices to assign each string a fitness value equal to the cost of the whole tour. As this function is to be minimized, that value must be inverted or, ever better, be subtracted from the fitness value of the worst individual in the population (whose new fitness will consequently be 0). An aspect to take in account is the apparition of scaling problems since, as the population evolves, differences between solutions tend to be smaller and even negligible in relation with the absolute values of them. So it is necessary to realize some kind of a normalization process in order to amplify that difference and guide more efficiently selective pressure.

**III.3.1.2 Representation function**

A first classification of the representation functions existing for the TSP partitions them into binary and non-binary functions.

Binary encoding fits to the most classical GA model. In this one, each node is represented by a segment of *n* bits (*n* is a natural number obtained from rounding up by excess log*m*, where *m* is the number of nodes), and a solution by an ordered sequence of such segments. This encoding has many problems.

First, if the number of nodes is not a power of 2, there exists segments without a corresponding node and, as a result, non-valid binary strings. Second, any operator working on this encoding should consider the segments corresponding to each node as *atomic* units; in any other case many non-valid strings would been generated. Although some positive results have been reported using this representation [(Michalewicz, 1992)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#MI92), it is usually less efficient and performs worse than other encoding functions.

Non-binary encodings use an alphabet with *n* symbols (*n* is the problem size) to represent solutions for the problem. There exist some options to encode: **path**, **adjacency** and **ordinal**.

**Path** encoding is the most intuitive one: every node is assigned a number (e.g. from 0 up to *n*-1) and solutions are represented by the ordered sequence of visited nodes.

**Adjacency** encoding uses each position in the string to represent an edge of the tour. So, if *j*th position of the chromosome contains the value *k*, the edge (*j,k*) belongs to the tour.

**Ordinal** encoding is the third option. Using it, each solution is represented by a list of *n* values, such that the *i*th position of it cannot contain a higher value than *n-i*, due to the fact that every gen points to a position within a stack from where visited nodes are progressively extracted. Initially, the stack contains all nodes in a fixed, predetermined, arbitrary order.

Basic features of the encodings above are detailed in [(Michalewicz, 1992)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#MI92) and [(Suárez, 1994)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#SU94). There is a third way to encode tours, *matrix encoding*, which has some varieties too. Since the size of matrixes grows according to the square of the problem size, they require a large amount of space. A description of these representations and their associated operators can be found in [(Michalewicz, 1992)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#MI92).

**III.3.1.3 Operators**

The next point to specify is the available operator set. It is very dependent of the encoding function used since some operators are more appropriate for a determined representation and others are not applicable to it at all. In order not to make this section too extensive, only operators for path and adjacency encodings are described. Operators for ordinal and binary representations can be found in [(Suárez, 1994)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#SU94).

First, crossover operators for adjacency encoding are described. As mentioned, a crossover operator must produce one or more offsprings combining information of, usually, two ancestors. A synergyc recombination of this information is the way to achieve improvements in the solutions provided by the algorithm. The most classical operators working on this representation are:

*Alternating-Edges Crossover*

*Subtour-Chunks Crossover*

*Heuristic Crossover*

For path encoding, the most notable operators are:

*Partially-Mapped Crossover or PMX*

PMX is an operator proposed by [Goldberg and Lingle (1985)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#GL85). It is designed to preserve many absolute positions from both parents. It works selecting two cutpoints in the first parent and copying the elements between them. This transfer also defines a set of mappings between the elements that have been copied and the elements in the corresponding positions in the second parent. Then, the rest of elements are copied in the positions they occur in the second parent. If one position is occupied by an element already copied from the first parent, the element provided by the mappings is considered. This process is repeated until the conflict is solved.

*Cycle Crossover or CX*

CX is an operator that was proposed by [Oliver *et al.* (1987)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#OSH87). It generates offspring in which every position come from one of the parents. Its functioning is based in the concept of *cycle*. A cycle is a minimal subset of elements such that the set of positions in which they appear is the same in both parents. This implies that it is possible to switch that subset from one parent to the other one while keeping a valid permutation. This operator copies the cycle that contains the first element of the first parent in the positions in which they occur in it, taking the rest of positions from the second parent.

*Order Crossover or OX*

There exist three different variants of OX.

The first variant of order crossover operator (OX#1) was first proposed by [Davis (1985)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#Dav85). It works selecting two cutpoints into the sequence, copying the elements between these points from one parent and preserving the relative ordering of the rest of elements in the second parent, starting after the second cutpoint and considering the sequence as a ring.

The second variant (OX#2) was proposed by [Syswerda (1991)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#Sys91). This operator selects some positions at random in the first parent and copies them into the offspring. The remaining positions are taken from the second parent, starting from the beginning and respecting their relative ordering.

Finally, the third variant (OX#3) was also proposed by [Davis (1991b)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#DAb91) and combines the features of the two operators above. As the first operator, two cutpoints are selected and the elements between them are copied. As the second operator, the rest of elements are copied from the beginning of the second parent respecting their relative ordering

The main problem of the operators above is the low percentage of ancestor's edges appearing in the offsprings, as they usually work with absolute positions of nodes instead of focusing on edges between them. This causes that an operator to build an offspring just with edges from the ancestors be defined (*Edge-Recombination Crossover*). However, path encoding is too poor for it, so it is completed with an auxiliary structure called edge-map. Detailed descriptions of this and all operators above can be found in [(Davis, 1991b)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#DAb91), [(Michalewicz, 1992)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#MI92) and [(Suárez, 1994)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#SU94).

On the other hand, there are mutation operators. They perform a secondary but decisive role in the global functioning of the algorithm. Without the renovation of the genetic stuff they inject into the population, selective pressure would guide crossover operator to produce a degenerate population. The most significant mutation operators for the path encoding are:

*Inversion*: two random points within the string are selected and the segment between them is inverted. This operator put in two new edges in the tour.

*Insertion*: a node is extracted from the tour and inserted in a random position of the string. This operator introduces three new edges in the offspring.

*Shift*: it is an extension of the later in which a segment is extracted. Three new edges are inserted in the tour.

*Reciprocal exchange*: the values contained in two random positions are exchanged, thus introducing four new edges in the string (or two if the positions to exchange are consecutive).

*Neighbour exchange*: it exchanges the contents of two consecutive positions within the string, so it may be considered as a restricted version of the reciprocal exchange or the inversion operator, in that cut or exchange positions are always consecutive.

There are not many references to mutation operators for adjacency encoding, so a path operator is frequently adapted. Descriptions of such adapted operators can be found in [(Suárez, 1994)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#SU94).

**This page was last updated on 21-Nov-97**

|  |  |
| --- | --- |
| **IV.** | **MODELS FOR THE PARALLELIZATION OF GENETIC ALGORITHMS** |

**Contents of this chapter**

**IV.1** Need of Parallelization

**IV.2** Fine and Coarse Grain Models

**IV.3** A Global Vision of the Existing Parallel GA Software

**IV.4** Object Orientation and Parallelism

**IV.3** Applications and Relationships with the Sequential Models

**IV.1 Need of Parallelization**

Sequential GAs have many succesfull applications to very different domains but there exists a number of drawbacks in their utilization, namely:

 They are usually inefficient.

 Evaluations are very time-consuming.

 Populations need to be large for a numerous number of problems.

In order to solve these shortcomings and also in order to study new models of higher efficiency and efficacy, parallel models of GAs and in general of EAs have been devised and used. Parallel GAs are not only an extension of the traditional GA sequential model but they represent a new class of algorithms in that they make a different search work. Thus we do not only expect a **P**arallel **G**enetic **A**lgorithm (PGA) to be more efficient but also to need a smaller number of evaluations of the target function.

Many different approaches to PGAs exist and they differ in many aspects. It follows one possible classification of parallel GAs attending to three concepts: the parallel model, the distribution of the population and the implementation. This is not the sole classification one can made for PGAs because it is very normal that the existing PGAs combine several caracteristics from these apparently disjoin models.



**Figure 1**. *One Possible Classification of Parallel GAs.*

**IV.2 Fine and Coarse Grain Models**

In the beginning of the parallelization of these algorithms the well-known **farming** method was used. In this way a central processor performs the selection operations while the associated slave processors perform the recombination, mutation and the evaluation of the fitness function. The last operation (evaluation) usually represent the most time-consuming load of the genetic algorithm.



**Figure 2**. *A Farmed GA.*

However, One of the most accepted classification for parallel GAs divide them attending to their parallel grain, that is, the relationship between computation and communication. Thus we can distinguish among **coarse** and **fine** grain models.

Typical **coarse** grain models consist in a set of parallel subpopulations that evolve independently from each other and that periodically exchange individuals among them. The topology for the exchanges is usually a ring or else some similar disposition. The subpopulations undergo exactly the same genetic operations on the same kind of genotype and the result is the best individual found in the overall population.



**Figure 3**. *A Distributed (Migration) GA.*

When dealing with a distributed GA some additional parameters need to be defined. If migration of individuals has to take place then the **rate** for the migrations and the **number of individuals** to migrate come into scene. We also need to decide which individuals are going to migrate. Finally the decision on the target subpopulation determines the global topology of the distributed system. Traditionally, the topology is a ring, one individual -the best or a random individual- migrates in every migration step and migrations take place at some predefined and periodic moments (althought some criteria for more controlled migrations have been considered in this field, usually performing migrations when some condition holds -the standard deviation falls below a given bottom limit, etc...-).

The migration scheme is very suited for a implementation in a network of workstations. The existing UNIX networks with TCP/IP communication protocols are the preferred for many researchers due to its relatively easy used and also to the existence of many communication tools: socket interface, PVM, etc...

On the other hand **cellular** (also known as massively parallel GAs) genetic algorithms use a spatial disposition for the individuals of one single population. Genetic operations take place in parallel for every node of the population but every individual interacts only with its neighborhood. Hence the selection, crossover and mutation take place by only considering the adjacent strings. The replacement policy destroys the considered individual by overwriting it with the newly computed string.



**Figure 4**. *A Massively Parallel (Cellular) GA.*

Cellular GAs have been implemented on multiprocessors due to the close similarities between the model and the physical disposition of CPUs. OCCAM and different versions of C have been used in the implementation. At present, multiprocessors machines like Transputers are being shifted to applications in which we need to execute a sequential algorithm with high speed. However cellular GAs on multiprocessors are not very frequent because the parallel market has plenty of local area network (cheaper, very fast, resources sharing, ...). One intermediate approach could be the simulation of a fine-grained GA in a network of workstations.

In cellular GAs, the isolation by distance property allows a high diversity and the selection pressure is alse weaker due to the local selection operator. The appearance of new species of solution in the grid (for example) and the refinement of existing solutions are both allowed and desired. The exploitation/exploration balance is then in many applications quite well developed by a parallel cellular GA.

**Figure 5**. *Evolution in a Parallel (Cellular) GA.*

**IV.3 A Global Vision of the Existing Parallel GA Software**

In this section we make a survey of the algorithms and software more readily accessible in this area. Let us begin with a temporal review of parallel GAs (PGAs). The most well-known cgpGAs and their main characteristics
sorted by date of referred appearance are:

**TABLE I.** OVERVIEW OF PARALLEL GAs BY YEAR

|  |  |  |  |
| --- | --- | --- | --- |
| **cgpGA** | **Ref.** | **Year** | **Main Characteristics** |
| PGA | [[55]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[55]) | 1987 | Generational cgpGA on an Intel iPSC hypercube (8 CPUs). Migrate the best. Dynamic Top. |
| dGA | [[68]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[68]) | 1989 | Distributed populations. Good results with 20% of population migrants every 20 generats. |
| GENITOR II | [[73]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[73]) | 1990 | Steady-State islands with ranked selection and reduced surrogate crossover |
| PGA | [[51]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[51]) | 1991 | Sub-populations in a circular ladder-like 2-D topology. Migrate the best, local hill-climbing |
| SGA-cube | [[23]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[23]) | 1991 | Made for nCUBE2. This is the parallel extension of the well known simple GA of Goldberg |
| PARAGENESIS | --- | 1992 | Made for the CM-200. This places one individual in every CPU |
| PeGAsuS | [[59]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[59]) | 1993 | Targeted for MIMD machines and written in a very high and flexible description language |
| GAMAS | [[56]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[56]) | 1994 | Uses 4 very heterogeneous species (islands) and quite specialized migrations and genotyp. |
| iiGA | [[44]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[44]) | 1994 | Injection island GA with hierarchical heterogeneous nodes and asynchronous migrations |
| SP1-GA | [[42]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[42]) | 1994 | 128 steady-state islands on an IBM SP1 machine of 128 nodes. 2-D toroidal mesh. mr=1 |
| DGENESIS | [[47]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[47]) | 1994 | Free topology, migration and policies for selection. Implemented with sockets and UDP |
| GALOPPS | [[30]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[30]) | 1996 | Very flexible. Implemented with PVM and comprising a large number of operators |
| GDGA | [[45]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[45]) | 1996 | Synchronous. Simulated on one processor. Generational. Fuzzy crossover and FP genes |
| CoPDEB | [[2]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[2]) | 1996 | Every island uses its own probabilities for mutation & crossover and specialized operators |

Of course this list is not complete, but it gives us a feeling of the evolution of cgpGAs over recent years. We will
offer more details about these and other algorithms in the following tables.

There has been a tendency towards making PGAs that run on clusters of machines paying a great attention to
their portability. Old systems like transputer machines do not support recent implementations. Also, new
developments in JAVA are appearing as a confirmation of these tendencies.

In order to complete our historical and application overview we give an original classification of sequential and parallel  GAs into three major categories [[59]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[59]) according to their specific objectives:

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

** Algorithm Oriented**: Based on specific algorithms. The source code is available in order to provide their 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 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 contain many different operators to help future applications.

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

 **Educational**: Used for introducing GA concepts to novice users (e.g. GA Workbench). The basic techniques to track executions and results during the evolution are easily managed.

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

We present some detailed tables with a classification of algorithms, and mark this classification with the above labels. Also, we use the PROP label for distinguishing between commercial and proprietary systems. We have also opened the contents of this table to non-cgpGAs to offer a better view of the global achievements of this class of evolutionary algorithms. See the following tables.

**TABLE II.** OVERVIEW OF 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]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[32]) | C | 64 T800 transp | Algorithm Specific | Single / No pop. | PAR | Local | --- |
| CoPDEB | [[2]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[2]) | C | UNIX | Algorithm Specific | Single / No prop. | PAR | Gen. | adamidis@it.teithe.gr |
| DGENESIS 1.0 | [[47]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[47]) | C | UNIX | Algorithm Specific | Single / No Prop. | PAR | Percentage | [ftp.aic.nrl.navy.mil](http://www.lcc.uma.es/~ccottap/semEC/cap04/ftp/ftp.aic.nrl.navy.mil) |
| ECO-GA | [[21]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[21]) | C? | UNIX? | Algorithm Specific | Single / No Prop. | PAR | Local | yuval@wisdom.weizmann.ac.il |
| EM 2.1 | [[69]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[69]) | C | DOS | Algorithm Library | Algrthm. Library | SEQ | Gen. / S. S. | [130.149.192.50]/  |
| EnGENEer | [[60]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[60]) | C | UNIX | General Purpose | Algrthm. Library | SEQ/par | Gen. / S. S. | --- |
| ESCAPADE 1.2 | [[35]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[35]) | --- | --- | Algorithm Specific | Single / No Prop. | --- | E. S. | 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]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[37]) | C | DOS | Educational | Algrthm. Library | SEQ | Percentage | camcon.co.uk |
| GAGA | [[8]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[8]) | C | UNIX | Algorithm Specific | Single / No Prop. | SEQ | --- | cs.ucl.ac.uk/darpa/gaga.shar |
| GAGS | [[48]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[48]) | C++ | UNIX & DOS | General Purpose | Algrthm. Library | SEQ | Gen. | <http://kal-el.ugr.es/GAGS> |
| GAlib | [[71]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[71]) | C++ | UNIX | Algorithm Specific | Algrthm. Library | SEQ | Gen. | lancet.mit.edu/pub/ga/galib242.tar.gz |
| GALOPPS 3.1 | [[30]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[30]) | C | UNIX & DOS | Algorithm Specific | Single / No Prop. | PAR | Gen. | isl.msu.edu/pub/GA |
| GAMAS | [[56]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[56]) | C? | UNIX | Algorithm Specific | Single / No Prop | PAR | Gen. | --- |
| GAME | [[66]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[66]) | C++ | UNIX & DOS | General Purpose | Algrthm. Library | SEQ/par | Gen. / S. S. | bells.cs.ucl.ac.uk/papagena/game |
| GAucsd 1.2 / 1.4 | [[61]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[61]) | C | UNIX, DOS ... | Algorithm Specific | Single / No Prop. | SEQ | Gen. | <http://www.aic.nrl.navy.mil/galist/src/GAucsd14.sh.Z> |
| GDGA | [[45]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[45]) | C | UNIX | Algorithm Specific | Single / No Prop. | SEQ/par | Gen. | lozano@decsai.ugr.es |
| GENESIS 5.0 | [[33]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[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]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[73]) | C | UNIX | Algorithm Specific | Single / No Prop. | SEQ-PAR | S. S. / Rank | [\*/genitor](http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/systems/genitor) |
| HSDGA | [[70]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[70]) | C | HELIOS | Algorithm Specific | Single / No Prop. | PAR | Local | --- |
| LibGA | [[18]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[18]) | C | UNIX | Algorithm Specific | Single / No Prop. | SEQ | Gen. / S. S. | [\*/libga](http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/systems/libga) |
| OMEGA | [[66]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[66]) | --- | --- | Application Orient. | Prop. | SEQ | --- | KiQ Limited (used in business applications) |
| OOGA | [[22]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[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](http://www.lcc.uma.es/~ccottap/semEC/cap04/%5B192.26.18.74%5D/pub/galist/src/ga/paragenesis.tar.Z) |
| PC/BEAGLE | [[59]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[59]) | --- | DOS | Application Orient. | Prop. | SEQ | S. S. | Pathway Research Ltd. (used in machine learning) |
| PeGAsuS | [[59]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[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](http://www.aic.nrl.navy.mil/galist/src%20/pga-2.5.tar.Z) |
| PGAPack | [[43]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[43]) | C | UNIX, ... | Algorithm Library | Algrthm. Library | PAR | Gen. / S. S. | <http://www.mcs.anl.gov/pgapack.html> |
| RPL2 | [[57]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[57]) | C | UNIX, ... | General Purpose | Prop. Library | PAR | Gen. / S. S. | <http://www.quadstone.co.uk/~rpl2> |
| SGA | [[26]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[26]) | C | UNIX | Algorithm Specific | Single / No Prop | SEQ | Gen. | [192.26.18.56] ... sgac\_94m.tgz |
| SGA-Cube | [[23]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[23]) | C | nCUBE 2 | Algorithm Specific | Single / No Prop | PAR | Gen. | [192.26.18.56] ... sgacub94.tgz |
| SPLICER 1.0 | [[54]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[54]) | C | UNIX & MAC | General Purpose | Algrthm. Library | SEQ | Gen. | galileo.jsc.nasa.gov |
| SUGAL 2.0 | [[38]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[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](http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/systems/tolkien) |
| XpertRule GenAsys | --- | --- | --- | Application Orient. | Prop. | SEQ | --- | Attar Software (used in design and scheduling) |

**TABLE III.** MEANING OF THE SYMBOLS IN TABLE II

|  |  |
| --- | --- |
| **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> |
| ** | pub/software/Evolution\_Machine/em\_tc.exe |

**TABLE IV.** CLASSIFICATION OF THE GENETIC SYSTEMS

|  |  |  |
| --- | --- | --- |
| **Application** | **Algorithm Oriented** | **Tool Kits** |
| **Oriented** | **Algorithm Specific** | **Algorithm Libraries** | **Educational** | **General Purpose** |
|   | ASPARAGOS |   |   |   |
|   | CoPDEB | EM 2.1 |   |   |
| EVOLVER 2.0 | DGENESIS 1.0 |   |   | EnGENEer |
|   | ECO-GA |   |   |   |
|   | ESCAPADE 1.2 |   |   | GAGS |
|   | GAGA | OOGA |   |   |
|   | GAlib |   |   | GAME |
| OMEGA | GALOPPS 3.1 |   |   |   |
|   | GAMAS |   |   | PeGAsuS |
|   | GAucsd 1.2 / 1.4 |   | GA Workbench |   |
|   | GDGA | PGAPack |   | RPL2 |
|   | GENESIS 5.0 - GENEsYs 1.0 |   |   |   |
| PC/BEAGLE | GENITOR I Y II |   |   | Splicer 1.0 |
|   | HSDGA |   |   |   |
|   | libGA |   |   | TOLKIEN |
|   | PGA 2.5 | SUGAL 2.0 |   |   |
|   | PARAGENESIS |   |   |   |
| XpertRule GenAsys | SGA |   |   |   |
|   | SGA-Cube |   |   |   |

**TABLE V.** DETAILS OF 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 |

**IV.4 Object Orientation and Parallelism**

[(Alba, Troya, 1997)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#AL97)

Traditional imperative languages like C or PASCAL endow the implementation with a high computational speed. However, more actual technologies such as object oriented design can have an outstanding impact on the quality of the implementation. Among the benefits of using the OO methodology for a PGA implementation, the following ones can be mentioned:

* Reusing the software is at present difficult: a software tool is either too specific and thus useless for other problems or it is too general and thus inefficient. OO can help in reusing software, in the fast building of prototypes and also in the security of this software.
* A common architecture of classes for a genetic system could be created and enriched by different people and experience. Comparisons should then be readily simple and meaningful.

An object oriented implementation of a PGA can also help experimenting and combining different granularities, topologies, operators, hybridization, competition or cooperation with other techniques and offers many other added facilities to create new models of evolution.

Two actual implementations that follow the OO methodology in C++ are described in this section. We call them dGANN and dGAME. The high parameterization of parallel GAs, their combinations, comparisons and development have been largely enhanced by using the OO design. We discuss a general methodology for building a parallel GA system with high flexibility and then present the two mentioned actual models.

**IV.4.1 *PGA Models and OO Implementation***

This section discusses the architecture of a PGA. For that purpose, some ideas on parallel models and the use of object orientation are presented.

From an algorithmic point of view, useful existing techniques must be examined for inclusion in the PGA. Hybridization and theoretical background are needed to improve existing models or proposing new PGAs.

The figure below represents the elements we should consider when developing an OO parallel GA (or EA).



**Figure 6.** *Proposal of working environment for a PGA software system.*

Two mixed levels are shown in the figure. The design level is related to the study of existing approaches, the concrete parallel model, how to perform hybridization and the need of evaluating the PGA on a real-world testbed.

On the other hand, the implementation level is related to internal details of the system. We propose an object oriented design for several reasons we will justify later in this section. The way of achieving parallelism is also important for the resulting PGA system.

We strongly believe that any parallel model should help to efficiently solve real complex problems. You can use almost any sequential or parallel system for the study of theoretical issues, but both efficiency and multiple ways of expressing one or more parallel models are required for addressing real problems . Changing or adding operators (static or dynamically during the search) should be made in a structured (and not in a problem-dependent) form.

Parallel algorithms do not usually run in massively parallel computers since LANs are much more popular. Some present types of LAN technologies as Fast Ethernet or ATM make communication delays as small as hard disk accesses. Hence massive parallelism should be simulated in the local network [(Maruyama, Hirose, Konagaya, 1993)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#MA93).

When simulating parallelism in one processor you must choose between having a process that encapsulates the parallel model directly or using threads (if possible, kernel threads). Sometimes a model could work as different UNIX processes running in the same machine, but might become very inefficient in a distributed environment.

**IV.4.2 *The Benefits of Using an OO Design***

The traditional aspect of a GA (see figure 7) can drastically change by means of an OO design and implementation.



**Figure 7.** *Traditional imperative pseudocode of a GA.*



**Figure 8.** *The shape and semantics of this GA differ from implementations built with imperative languages. For example, explicit initialization of the GA could be helpful for controlling its execution in a parallel platform.*

With practically the same amount of effort we can both specify an imperative pseudocode or else a concrete-working code for a problem in terms of object orientation. The last solution is much more flexible and rich in the data structures it uses and also in the performed operations (as well as in future extensions to other problems).

Object orientation is not just a new style for implementations. It represents a new form of conceiving the design of a software tool.

Thus, not only we can express in a new manner a PGA but we can also add new features such as new schemes of evolution, genotypes, lists of operators, parallel models and parameters in almost a straightforward way.

**IV.4.3 *The Two Proposed Systems***

In this subsection we want to propose two separated systems that incorporate the main ideas we outlined in the previous sections on PGAs and OO.

In order to create objects suited for structured addition of operators you need to completely separate the class of *individuals* and the class of the *operator pool*. The operator pool is a list of operators that can dynamically accept or reject new operators, and of course change their list of parameters. The idea of an ordered list of operators to be applied each one with its own list of parameters (and not just its application rate) can be very helpful in controlling and enhancing the PGA performance.

The individuals also must be separated from their contents in the sense that the kind of genotype must be a different class in order to allow an independent manipulation. Hence a problem class must exist to separate the full software system from the actual problem we are trying to solve with it.

Finally we need communication classes for transferring data among processors and we need also some classes for expressing parallel interactions among GAs. A central monitor is normally needed in order to deal with the user (get global and particular statistics and status, kill a process, save and load populations on line, etc...). In some models it should be interesting the central monitor to make also algorithmic work for the PGA.



**Figure 9.** *Proposal of a hierarchy of dGA classes (dGANN).*

Figure 9 shows the class hierarchy for the dGA system we call dGANN. This proposal deals with fault tolerance in the sense that if a genetic process dies the system is able to keep on running. We propose asynchronous exchanges of information (like in [(Gorges, 1989)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#GOR89) or [(Munetomo, 1993)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#MU93)), because this will ease the fault tolerance aspect (synchronous communication would be too slow).

A socket interface has been initially considered the faster and more compatible way of implementing a distributed PGA. Classes for exchanging messages are needed and also for controlling the integrity of the system during execution. New genotypes can be easily added to this class hierarchy.

In order to have a good general parallel model we think that the implementation could take into account (of course the experience of other existing models and) the benefits of using C++. You can use another OO languages as Eiffel, Ada or Smalltalk but C++ is the most widely accepted.

Despite its advantages, C++ do not provide parallel support. Thus if we want to parallelize an OO model we need some kind of communication services for the implementation. Parallelization is specially easy and flexible with OO.

The drawback is that cellular GAs cannot be easily derived from this hierarchy since the object that represents one individual should run in parallel with the rest.

Cellular GAs (or mpGAs) have many points in common with distributed GAs (dGAs):

* They both use some kind of spatial topology: dGAs usually a ring and mpGAs usually locate individuals in a grid.
* They both make parallel exchanges of information among neighbors (neighbors are subpopulations of N individuals in dGAs or else just single individuals in mpGAs).
* The individuals of a traditional population can be viewed as a full-connected cellular topology.



**Figure 10.** *The dGAME model. Merging of models can yield more robust searches. There exist two levels of search: a dGA of mpGAs.*

In the above example the low monitors exchange individuals in a ring topology (much like a migration dGA scheme). Each L-monitor controls a population with a given cellular (mpGA) disposition of individuals.

With these considerations we are ready to think in a unified parallel model that can be implemented in the same software system and that additionally opens new questions relating the merging and cooperation among different submodels. This is good if we can capture in a given parallel model the advantages of the combined submodels (trying to minimize their associated drawbacks). Our dGAME extension of the GAME [(Stender, 1993)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#ST93) environment is the first step in this sense.

Besides these new ideas, cooperation or competition among different kind of GAs or even different algorithms can be achieved. Such an OO distributed GA could deal with most of requirements of existing problems.

Most of these concepts can be found in our dGAME environment. An OO design, combinations of topologies and merging of arbitrary dGA-mpGA models are the more important characteristics of dGAME.

**IV.5 Applications and Relationships with the Sequential Models**

From a different point of view we present in table IV a set of representative applications of PGAs in order to
show the wide spectrum of their successful applications (i. e. robustness).

**TABLE VI.** SOME APPLICATIONS OF PARALLEL GAs

|  |  |
| --- | --- |
| **Reference** | **Application Domain** |
| [[7]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[7]) | Parallel training of artificial neural networks. Hybridized and pure cgpGAs |
| [[19]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[19]) | Synthesis of VLSI circuits |
| [[31]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[31]) | Function optimization |
| [[42]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[42]) | Set partitioning problem |
| [[44]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[44]) | Graph partitioning problem |
| [[49]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[49]) | Constraint Optimization, reordering problems, ... |
| [[51]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[51]) | Traveling salesperson problem (TSP), function optimization |
| [[53]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[53]) | Distributing the computing load onto a set of processing nodes |
| [[56]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[56]) | The file allocation problem, XOR neural network, sine envelope sine wave function |
| [[66]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[66]) | Systems modelling, protein tertiary structure prediction and two-dimensional bin packing problems |
| [[68]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[68]) | Walsh polynomials |
| [[73]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[73]) | Optimization of the connection weights of neural networks (XOR, bin-adder, ...) and function optimization |

There exist many other applications in very interesting domains such as frequency assignment problems, market
predictions, game theory, filters, graphic transformations, etc. Some of these applications motivated the need for
new classes of genotypes that can be thought of consisting in strings of arbitrary symbols. A further extension led to the mentioned *genetic programming* paradigm [[40]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[40]) in which the individuals are *trees of symbols* representing computer programs, fuzzy controllers, or many other kinds of high-level concepts. This is a new and important research area in
which PGAs are developing quickly. Also new genotypes and operators are being developed for dealing with
constraint problems and combinatorial optimization.

Besides that, the importance of parallel and distributed GAs is growing due to recent studies in which no parallel
hardware is needed but still the search is enhanced due to neighborhoods similar to [[32]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[32]), [[46]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[46]) or [[51]](http://www.lcc.uma.es/~ccottap/semEC/cap04/cap_4.html#[51]).

**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". *Informática y Automática*, 30(3), pp 42-60. September, 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, D. Fogel, Z. Michalewicz (eds.) *Handbook of Evolutionary Computation* (Oxford University Press). 1997.
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. 1985.
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. "Parallel Genetic Algorithms in Distributed Memory Architectures". *Master 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. D. E. Goldberg. "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. D. E. Goldberg, 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, 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 ESCAPADE 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, 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 for the Execution of Distributed Genetic Algorithms". *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. Schraudolp, 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, B. Manderick. "A Massively Parallel Genetic Algorithm". *Proceedings of the 4th International Conference on Genetic Algorithms*, R. K. Belew, 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, B. Manderick (eds.), North-Holland, Amsterdan, pp 155-164. 1992.
71. M.Wall. "Overview of Matthew’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. Starkweather. "GENITOR II: a Distributed Genetic Algorithm". J. Expt. theor. Artif. Intelligence 2, pp 189-214. 1990.

**This page was last updated on 6-May-98**

|  |  |
| --- | --- |
| **V.** | **DIRECTLY RELATED GA TECHNIQUES** |

**Contents of this chapter**

**V.1** Adaptive Parameterizations

**V.1 Adaptive Parameterizations**

[(Davis, 1989)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#DA89)

**V.1.1 *Introduction***

Running a genetic algorithm entails setting a number of parameter values. Finding settings that work well on one problem is not a trivial task; if poor settings are used, a genetic algorithm performance can be severely impacted. This section deals with a new technique for setting the probabilities of applying genetic operators during the course of a run. The technique involves *adapting* the operator probabilities based on their observed performance as the run take place. In the section that follows, we summarize prior work in determining reasonable operator probabilities, describe experiments measuring the performance of the new technique, and describes a use of the technique in its first real-world application.

**V.1.2 *Finding Good Parameter Settings Can Be Hard***

The problem dealt with in this paper is that a given genetic operator will be applied during reproduction. Setting operator probabilities correctly can be very important when one has limited resources with which to solve a problem or when one is comparing the performance of a genetic algorithm with the performance of other types of algorithms on the same problem. Two useful strategies have been employed by researchers in the genetic algorithm field to find good operator probabilities. One was contained in Ken DeJong's thesis work [(DeJong, 1975)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#DJ75); the other was described by John Grefenstette in [(Grefenstette, 1986)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#GR86).

The problem DeJong and Grefenstette were attacking was that of determining robust parameter settings for population size, operator probabilities, and evaluation normalization techniques. The problem of adapting genetic algorithm parameters in general was simplified, in that the only chromosomal representation technique used was the bit string, and only two operators were considered - mutation and crossover. This is a simpler problem that of finding robust parameters for genetic algorithms across all representations, across the range of all possible operators, and so forth. Yet, even stripped down to the simple form just given, the problem is not easy to solve.

A first difficulty is that even in this stripped-down form, the problem is not precisely specified. We are asking for good values for the parameters, but measuring how good such values are is non-trivial, since the range of potential applications of genetic algorithms using binary representations is of infinite size. DeJong solved this difficulty by simplifying the problem more: he used a *test suite* of function optimization problems compiled by researchers in the field of function optimization theory to stand in for the full range of possible domains. DeJong's choice of functions in the test suite was a good one.

A second difficulty in solving this optimization problem given a test suite is that the evaluation function is noisy. Genetic algorithms are stochastic, and the same parameter settings used on the same problems by the same genetic algorithm generally yield different results. A consequence of this fact is that it can take a tremendous amount of computer time to find good parameter settings across a number of problems. DeJong's research was of immense benefit to people using genetic algorithms because it provided them with parameter values that were robust, in the sense that they had been validated on a number of different problems of different types.

The problem of noise in the evaluation function meant that DeJong was forced to derive his parameter values by hand. Several years later, however, as computing resources had become cheaper , Grefenstette derived new values for the parameters in question by using a genetic algorithm to do so. Grefenstette approach was to evolve parameter values encoded in binary on a chromosome, evaluating each chromosome by assigning the values it encoded to a genetic algorithm that was run on each problem in DeJong's test suite. The result, at greater cost in computing resources, was a new set of parameter settings that out-performed DeJong's settings.

Given this history of difficulty in optimizing six parameters involved in a genetic algorithm with a simple representation format and two genetic operators, it is understandable that many researchers have shied away from considering representations that are not binary and operators that are different from the two that have been parameterized at such expense. But this raises a third problem suggested by the research described above: a great many real-world optimization problems appear ripe for solution by genetic algorithms, yet the binary representation appears ineffective or inefficient for them. Further, operators other than binary crossover and binary mutation appear to contribute to good performance in those domains. A second, and potentially more important, problem, then, is that of parameterizing a genetic algorithm that differs from the type studied so thoroughly by researchers in the field.

**V.1.3 *The Nature of the Genetic Algorithm Used Here***

The genetic algorithm that has been used to test adaptive operator probability routines had the following characteristics. It was steady state; that is, instead of carryng out generational population replacement, children were inserted after each reproduction event. The system used a ranking technique for normalizing the evaluations of each individual, and the normalized evaluations were the result of assigning exponentially decreasing values to each individual in ranked order. Thus, given a value of .95 with which to produce evaluations, the best individual in the population was given 1000 x .95 = 950, the next was given 950 x .95 = 902.5, and so on, with the exception that no individual was given a normalized evaluation less than 1. The parameter that determined the normalized evaluations was interpolated from .95 to .3 over the course of the run, with an interpolation interval of 50 children. Deletion of population members was done by removing the last member of the population. The population size was 100. the total number of individuals evaluated in each run was 1000.

The system was operator-based: that is, each reproduction event was the result of asking a single operator to apply itself. Each reproduction event used exactly one of the following operators:

**GUARANTEED-UNIFORM-CROSSOVER.** Create two children from two parents by deciding randomly, field by field, which child receives the corresponding value from which parent. Do not use a child that is identical to either parent.

**GUARANTEED-AVERAGE.** Create one child from two parents by averaging their fields. Choose randomly which integer to use if the average lies between two integer values. Use the child only if it differs from both parents.

**GUARANTEED-MUTATION.** Create a child from one parent by passing down the parent's fields and, with 10% probability, replacing each value by a random value between 0 and the maximum allowed value. Use the child if it differs from the parent.

**GUARANTEED-BIG-CREEP.** Create one child from one parent by passing down the parent's fields and, with 20% probability, altering a field by replacing it with a value that is 1, 2 or 3 units above or bellow the field value. Select the amount and the direction of the creep randomly. Use the child only if it differs from the parents.

**GUARANTEED-LITTLE-CREP.** Create one child from one parent by passing down the parent's fields and, eith 10% probavility, altering a field by replacing it with a value that is 1 unit above or bellow the field value. Select the direction of creep randomly. Do not go bellow 0 or the maximum allowable value in creeping values. Use the child only if it differs from the parent.

The term "guaranteed" is a local check that an operator is not producing a child that is a copy of its parents.

**V.1.4 *The Adaption Mechanism***

The idea that one should adapt one's genetic algorithm during the course of a run is not a new one. C. Shaefer [(Shaefer, 1986)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#SH86) has built an approach to numerical optimization around this idea in his ARGOT program. The thing adapted in ARGOT is the representation itself, and pursuing this idea has led Shaefer to some interesting results. Shaffer and Morishima [(Shaffer, Morishima, 1987)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#SH87) have investigated encoding crossover points on chromosomes, so that information controlling crossover locations evolves during run. Darrell Whitley [(Whitley, 1987)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#WH87) has investigated the adaptation of the reproductive fitness of a parent based on performance of its offspring. The idea described here is that one adapts the probability that a genetic operator will be applied in reproduction based on the performance of *the operator's* offspring.

The first intuition underlying the adaptation mechanism is that an adaptive mechanism should alter the probability of applying an operator in proportion to the observed performance of the individuals created by that operator in the course of a run.

A second intuition underlying the mechanism is that purely local measures of performance are not suficient. One must reward an operator that produces a good child, but one must also reward an operator that set stage for this production. This intuition is grounded in the sort of situation that often happens toward the end of a run, whe one's population is mostly converged.

This intuitions were implemented in the following way. Whenever a new member was added to the population, a pointer was established to its parent or parents, and a pointer was established to the operator that had created the new member. At the same time, a check was made to determine whether the new member was better than the current best member of the population. If so, the amount that it was better was stored with it as its "local delta". Each of the operators above was given an initial probability. Periodically, these weights were altered, using the following parameters: *W* (the adaption window) - how far back to consider individuals in computing delta; *S* - the amount of weight that will be shifted each tine adaption is done; *I* - the interval between adaptations; *P* - the proportion of delta that is passed back to an individual's ancestors when computing delta; *M* - a limit on the number of generations that delta is passed back.

The technique for computing the delta associated with each operator is as follows. First, each of the last *W* individuals added to the population passes *P* times its local delta back to its parents. This quantity is shared among the parents. Each of those parents passes *P* times the quantity it received back to its parents, and so on until delta has been passed back *M* generations or until an ancestor outside the adaption window is about to receive delta. Now each of the last *W* individuals in the population computes its *delivered delta* by summing its local delta and its inherited delta. The delta for an operator over an adaptation interval, given individuals *I1...In* that were created by the operator, is the sum of the derived daltas for *I1* through *In<* divided by *n*. That is, the delta for an operator over an interval is the sum of the derived deltas of the individuals it produced over the interval, divided by the number of individuals it produced. If an operator produced no individuals, its delta is 0.

If we have *j* operators, a current vector of probabilities *V* of length *j*, and a vector *D* of derived deltas associated with each operator, we carry out the adaptation of the probabilitiesin *V* in the following way. If there is no delta over the interval, the probabilities are not changed. Otherwise, we form *V'*=((100-*S*)/100)\**V*, (*V'* is *V* with S% proportionally removed). Next, we form *D'*, the result of normalizing *D* so it totals *S*. Now we set the new vector of operator probabilities *V''*=*V'*+*D'*.

**V.1.5 *Concluding Comments***

The mechanism described here has several features that should be noted. It allows rapid parameterization of operator probabilities across the range of potential genetic algorithms and operator set. As described here, it is tailored to a steady state reproduction scheme. It would not be literally applicable to problems with noisy evaluation functions. Further, Davis believes that the feedback rate would be greatly damped using generational replacement. Thus, futher work will be required to adapt it to generational genetic algorithms and noisy-reinforcement domains.

Using an adaptive mechanism leads to performance slightly inferior to that carefully-derived iterpolated parameter settings on th test problem tha Davis performed. Observation of those cases in which th adaptive mechanism does badly shows that there are many in which an operator's probability varies wildly from the average trajectories.

This suggests that one could derive parameters for interpolating values by doing a manipulation of th adaptive trajectories - by doing a best fit of a line to the adaptive trajectories, for example. The adaptive mechanism uses strictly local information to modify operator probabilities. The interpolation mechanism uses more global information. It appears that we could combine both mechanisms to out profit.

The most significant parameter impacting solution of the test problem was not the relative probabilities of the operators. Rather, it was a parameter having to do with the selection of parents. This parameter, too, can be adapted usefully over the course of a genetic algorithm run.

The mechanism reported here is in its infancy, but it has already provided invaluable assistance in several real-world applications of genetic algorithms to problems with representations and operator sets that are widely different from the representation and operator sets studied by DeJong and Grefenstette. There is no good deal of work to do in order to make the mechanism more robust, to study its properties further, to run it using binary representation on the standard test suite, and so forth. Combined with techniques for adapting the *other* parameters in a genetic algorithm, it's hopped that it will help us to produce quick and effective applications to real-world problems.

**This page was last updated on 20-Apr-98**

|  |  |
| --- | --- |
| **VI.** | **GENETIC PROGRAMMING** |

**Contents of this chapter**

**VI.1** Background

**VI.2** GP for the Generation of Fuzzy Rules

**VI.1 Background**

[(Sánchez, 1996)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#SA96)

In areas such as artificial intelligence, machine learning, or symbolic processing, we find many problems whose resolution can be considered as the search of a computer program, inside a space of possible programs that produce some desired outputs from the inputs. This search should be carried out in such a way that the searched program be the more adequate to the problem that we are considering.

The genetic programming (GP) paradigm provides us the appropriate framework to develop this type of search in an efficient and versatile way, since it adapts to any problem type, i.e., it is very robust.

The paradigm of genetic programming is based on the principle of survival of fittest (C. Darwin). Starting from an randomly-generated initial population, it evolves populations following this principle. The new individuals are a product of genetic operations on the current population's better individuals. In connection with the genetic algorithms (GAs), GP shares with these the philosophy and the characteristics of being heuristic and stochastic.

**VI.1.1 *The GP Paradigm in Machine Learning***

Inside the area of the machine learning, we find several paradigms focused toward the resolution of problems. In each paradigm the used structures are different:

**Connectionist Model**: the solution to the problem is given as a group of real-valued weights that indicate if the value of the signal that goes by a certain connection of a neural network is amplified or diminished.

Application field: NEURAL NETWORKS

**Evolutionary Model**: the solutions are fixed-length strings (in the classic model). Each chromosome represents a possible solution to the problem. A conventional genetic algorithm is applied to obtain the best solution (or a good enough solution) among all possible solutions.

Application field: GENETIC ALGORITHMS

**Inductive Model**: according to this paradigm the solutions to a certain problem are given by decision trees. Each one of these trees classifies each instance of the problem in classes, for which a possible solution exists.

Application field: CLASSIFFIER SYSTEMS

Each one of the mentioned models can be more or less effective solving a certain problem type. The approach that is used to determine the efficiency of a method is, in the first place, the flexibility to adapt to several types of problems, and in second its easiness to represent the solutions to this problem in a natural and comprehensible way for the user. Computer programs offer flexibility:

to perform operations with variables of different types.
to carry out operations conditioned to the results obtained in intermediate points.
to carry out iterations and recursions.
to define routines that can be used later on.

This idea of flexibility includes the concepts of flexibility in the size, the form and the structural complexity of the solution. That is to say, the user should avoid stating any type of explanation or previous imposition on the size or form of the solution. The true power of GP resides in its capacity of adaptation to the problem, for what the considerations on the size, the complexity or the form of the solution should emerge during the own resolution process.

The importance of the representation of the solutions resides in that the genetic algorithms manipulate the structure of this representation directly and not the own solution.

String-representations do not directly provide the hierarchical structure of programs. Also, the adaptation of the form, size and complexity of individuals becomes very difficult. Therefore, GP has spread toward more complex representations that contribute the needy flexibility. Smith introduces the strings of variable length and IF-THEN-ELSE components [(Smith, 1980)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#SM80).

Genetic algorithms are used in a wide range of applications thanks to the adaptation of their evolutionary structures. For example as classification systems, as the system designed by Wilson for the classification of boolean functions [(Wilson, 1987)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#WI87). Goldberg introduced the Messy genetic algorithms that handle populations of individuals represented by strings of characters of variable length [(Goldberg, 1989b)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#GO89b).

GP uses programs like individuals, but the division of opinions appears with its implementation. Cramer uses a parse-tree-like representation of the code, and defines suitable operations (for example, exchange of subtrees for recombination)[(Cramer, 1985)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html%22%20%5Cl%20%22CR85). Fujiki and Dickinson have developed a system that is based on the generation of programs that use simple conditional sentences of LISP (COND) [(Fujiki, Dickinson, 1987)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#FU87).

As we have already seen, the resolution of many problems can be modelled as an evolutionary process in which the fittest individual survive. The simulation of this evolutionary process begins with the aleatory generation of an initial population, composed by computer programs that represent the individuals. These programs are generated starting from the group of functions and terminal elements that adapt better to the problem to solve. In most cases, the election of the group of terminal and non terminals (functions) is critical to make the algorithm works properly.

For example, if we want to generate complex mathematical functions it is interesting to introduce in the group of non terminals such functions as sines, cosines, and so on. For graphic applications, it is usually quite normal to introduce primitive of the type Line, Circle, Torus that graphically represent figures in diverse ways.

The appropriateness of each program is measured in terms of how of well it performs in the environment of the particular problem. This measure is denominated fitness. A program is generally evaluated in different representative cases, and the final measure of its fitness will be an average of all the measures (one for each case). Usually, the result of the genetic algorithm is the best individual generated in generation *n*, being *n* the maximum number of generations.

GP performs, in a genetic way, computer programs executing the three following steps:

|  |
| --- |
| **1)** Randomly generate the initial population, using the group of terminals and functions for the considered problem.**2)** Iteralively carry out the following steps until the termination criterion is satisfied: **a)** Execute every individual's program and assign it a fitness value, according to its ability to solve the problem.**b)** Create a new population applying genetic operators on the selected individuals: **i)** Reproduction.**ii)** Crossover.**3)**Take the best solution of the final population. |

**Figure 1**. *Steps of a GP.*

**VI.2 GP for the Generation of Fuzzy Rules**

[(Cotta, Alba, Troya, 1996)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#CO96) [(Alba, Cotta, Troya, 1996)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#AL96)

In this section, we describe an application of GP in which individuals (trees) are not computer programs in a traditional sense. Every tree will represent a set of fuzzy rules intended to solve a control problem (the cart centering problem -Figure 2-, described and analytically solved in [(Koza, 1992)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#KO92)). The underlying target programming language is neither C, PASCAL nor LISP. We use a fuzzy interpreter in order to compile and run the resulting fuzzy rules on our specific problem.



**Figure 2.** *The Cart Centering Problem.*

**VI.2.1 *Fuzzy Logic Controllers***

A Fuzzy Logic Controller (FLC) is a rule-based system that tries to incorporate the flexibility of human-decision making by means of the use of fuzzy set theory [(Lee, 1990)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#LE90). The rules of an FLC incorporate fuzzy linguistic terms (e.g. temperature is warm) described by membership functions. These functions are intended to represent human expert's conception of the linguistic terms, thus giving an approximation of the confidence with which a precise numeric value is described by a linguistic label.

Rules take the form IF [conditions] THEN [actions], where conditions and actions are linguistic labels applied to input and output variables respectively (e.g. IF temperature IS hot THEN refrigeration IS high). A set of such fuzzy rules constitutes the fuzzy rule-base of the FLC. The system uses this rule-base to produce precise output values according to actual input values. This control process is divided into three stages:

* fuzzification: calculate fuzzy input, i.e. evaluate input variables with respect to the corresponding linguistic terms in the condition side.
* fuzzy inference: calculate fuzzy output, i.e. evaluate activation strength of every rule and combine their action sides.
* defuzzification: calculate actual output, i.e. convert fuzzy output into a precise numerical value.



**Figure 3.** *Phases of fuzzy interpreter.*

The fuzzy interpreter used in this work performs fuzzification via triangular membership functions, uses the min intersection operator, the maxmin method for fuzzy inference and the centroid procedure for defuzzification (for comparation purposes with [(Thrift, 1991)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#TH91)).

**VI.2.2 *The Cart Centering Problem***

In this section a description of the cart centering problem will be made. Next, it will be shown how the principles of fuzzy control can be applied to this problem, outlining an ad-hoc solution.

A cart with mass m can move along an infinite one-dimensional frictionless track. The problem consists in centering the cart, in minimal time, by applying an external time-variant force (of maximal magnitude *F*) so as to accelerate the cart in either the positive or the negative direction of the track.

This problem can be further formulated as calculating a force *F(t) = F[x(t), v(t)]* in order to place the cart in position *x(tf)=0* with velocity *v(tf)=0* in minimal *tf*.

The exact time-optimal solution consists in applying a force *F(t)=F* when
and *-F* otherwise.

This system can be simulated using Euler's method considering time steps ():


As an optimal control problem, the cart centering problem is well-suited for fuzzy control. In order to develop an FLC for this system we take several steps:

1. Determation of the condition (input) vbles.-> position *x(t)* and velocity *v(t)*.
2. Definition of the membership functions representing the fuzzy sets over these inputs (5 partitions: NegLarge/NegSmall/Zero/PosSmall/PosLarge).
3. Identification of action (output) variables -> the force *F(t)* applied to the cart. We use the same fuzzy sets above to describe this output variable.
4. Production of the rule-base. This set of rules might have been designed by a human expert.



**Figure 4.** *Fuzzy sets for position, velocity and force.*

**VI.2.3 *A Hand-Made Fuzzy Solution***

We can define a human-expert conception of a solution as follows:

*The applied force must almost always be directed towards the origin (since we want to place the cart in the 0 position) and, just when we are close to that point, we must reverse that force to stop the cart*

The associated FLC rule-base can be expressed as:

1. **IF** *pos* **IS** PL **THEN** *for* **IS** NL
2. **IF** *pos* **IS** PS **THEN** *for* **IS** NL
3. **IF** *pos* **IS** ZE **AND** vel **IS** PL **THEN** *for* **IS** NL
4. **IF** *pos* **IS** ZE **AND** vel **IS** PS **THEN** *for* **IS** NL
5. **IF** *pos* **IS** ZE **AND** vel **IS** NS **THEN** *for* **IS** PL
6. **IF** *pos* **IS** ZE **AND** vel **IS** NL **THEN** *for* **IS** PL
7. **IF** *pos* **IS** NS **THEN** *for* **IS** PL
8. **IF** *pos* **IS** NL **THEN** *for* **IS** PL

The FLC defined by the membership functions and the rule-base described above is not the optimal one. It can be improved by means of a learning technique that modifies the rule set or the interpretation of the linguistic terms. Genetic algorithms have been used for that purpose in both ways [(Thrift, 1991)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#TH91), [(Karr, 1991)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#KA91), [(Lee, Takagi, 1993)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#LT93), [(Feldman, 1993)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#FE93). We propose the use of GP to produce rule bases that can be used as a starting point for further refinements by a human expert.

**VI.2.4 *Advantages of using Genetic Programming***

No analytical knowledge is needed and we can still get accurate results.

If we encode fuzzy sets in the genotype we can generate new -more suited- fuzzy sets to describe precise and individual membership functions. We can do it by means of the intersection and/or union of the existing fuzzy sets.

Every component of the resulting GP rule-base is relevant in some way for the solution of the problem. Thus we do not encode null operations that will expend computational resources at runtime.

This approach does scale with the problem size. Some other approaches to the cart centering problem use a GA that encodes NxN matrices of parameters. These solutions work bad as the problem grows in size (i.e. as N increases).

With GP we do not impose restrictions on how the structure of solutions should be. Also we do not bound the complexity or the number of rules of the computed solution.

**VI.2.5 *Genetic Programming of Fuzzy Logic Controllers***

**1. Encoding a Rule-Base**

As stated above, the rule-base of the FLC is a list of IF-THEN rules. Each of them can be easily represented as a binary tree: the root being an IF node and left and right branches representing the condition and the action sides respectively. Likewise, both conditions and actions can be expressed as trees. On the one hand, a variable paired with a fuzzy set can be represented as a tree with an IS root-node, the variable name in the left branch and the fuzzy set in the right branch. On the other hand, a conjunction of such terms can be expressed as a tree with an AND root-node and two branches representing nested conjunctions or pairs (variable, fuzzy set). Figure 5 shows an example.



**Figure 5.** *A syntactic tree and the rule it represents.*

|  |
| --- |
| BNF GRAMMAR OF TREES |
| <TREE><RLIST><IF><COND><L\_IS><L\_AND><ACT><R\_IS><R\_AND><SET><VBLEIN><VBLEOUT> | ::= **EL** | <IF> | <RLIST>::= <TREE> <TREE> **RLIST**::= <COND> <ACT> **IF**::= <L\_IS> | <L\_AND>::= <VBLEIN> <SET> **LEFT\_IS**::= <COND> <COND> **LEFT\_AND**::= <R\_IS> | <R\_AND>::= <VBLEOUT> <SET> **RIGHT\_IS**::= <ACT> <ACT> **RIGHT\_AND**::= **NL** | **NS** | **ZE** | **PS** | **PL**::= **VEL** | **POS**::= **FOR** |

Likewise, both conditions and actions can be expressed as *trees* with an IS root-node, the (in/out) variable name in the left branch and the fuzzy set in the right branch. A conjunction of such terms can be expressed as a tree with an AND root-node and two nested conjunctions branches.

**2. Add types to the Trees**

The flexibility of our tree representation makes crossover a very powerful tool since it allows interchange at several levels: rules, terms or even variables and fuzzy sets. Since traditional crossover may produce nonsense rules we can:

*Define closed functions*, so the ill-defined rules could be interpreted in some given way, e.g. computing fixed values for unexpected symbols.

*Repair the tree*, deleting (adding) incorrect (new) subtrees.

*Perform a restricted crossover*, i.e. resulting trees will be correct.

Since the first option will produce a high number of useless trees (dummy rules) and the second one is *computationally expensive* and will produce *offsprings very different* with respect to the parents, we have chosen the third option.

Crossover selects one random subtree from every parent. The selection of the root nodes of the two subtrees is forced to have the same type.

|  |  |
| --- | --- |
| **TYPE** | **SYMBOLS** |
| *Type I* | { IF, RLIST, EL } |
| *Type II* | { AND, IS } |
| *Type III* | { ... variable names ... } |
| *Type IV* | { ... fuzzy sets ... } |

**3. Refine types to keep consistency**

The presented types will lead to the creation of syntactically correct but semantically incorrect trees => a rule could be spawned in which output variables appear in the condition side and/or input variables appear in the action side (e.g. IF *for* IS NL THEN *pos* IS ZE).

If this kind of situations were allowed, the GP system would waste valuable resources in evaluating and storing such useless rules. To avoid this problem, we define two subtypes for both types II and III.

Every subtype can be seen as the *variety* of the corresponding type that appears in either the condition or the action side.

|  |  |  |  |
| --- | --- | --- | --- |
| **TYPE** | **MEANING** | **SUBTYPES** | **MEANING** |
| LIST | List of rules | - | - |
| COND\_ACT | Condition or Action | CONDITIONACTION | To separate branches of an IF structure |
| VBLE | Variable | VBLE\_INVBLE\_OUT | Input variableOutput variable |
| SET | Fuzzy Set | - | - |

**4. Conclusions**

The GP design of fuzzy logic controllers is a more flexible and natural way that traditional GAs since no a priori constraints are posed on the shape of the solution (in fact this shape also evolves): the complexity of the solutions are not bounded by the algorithm.

By including fuzzy sets in the trees we can also evolve them through their union and intersection, thus refining initially (hand-crafted) user proposed fuzzy set.

A typed system that defines the composition of the solution trees allows to keep strings feasibility at runtime at a low cost and prevents the GA of useless wanderings along the search space => more complex problems can be tackled.

The GP solution clearly outperforms intuitive solutions and is competitive with other genetic-based solutions regarding the time to dock the cart and in keeping a low level of complexity and the number of fuzzy rules.

**This page was last updated on 21-Nov-97**

|  |  |
| --- | --- |
| **IX.** | **EC RELATED TECHNIQUES** |

**Contents of this chapter**

**IX.1** Tabu Search

**IX.1 Tabu Search**

Tabu search (TS) has its antecedents in methods designed to cross boundaries of feasibility or local optimality normally treated as barriers, and systematically impose and release constraints to permit exploration of otherwise forbidden regions. Early examples of such procedures include heuristics based on surrogate constraint methods and cutting plane approaches that systematically violate feasibility conditions.

Webster's dictionary defines *tabu* or *taboo* as 'set apart as charged with a dangerous supernatural power and forbidden to profane use or contact...' or 'banned on grounds of morality or taste or as contituting a risk...'. Tabu search scarcely involves reference to supernatural or moral considerations, but instead is concerned with imposing restrictions to guide a search process to negotiate otherwise difficult regions. These restrictions operate in several forms, both by direct exclusion of certain search alternatives classed as 'forbidden', and also by translation into modified evaluations and probabilities of selection.

The purpose of this section is to introduce some of the fundamental points that characterize tabu search.

The philosophy of tabu search is to derive and exploit a collection of principles of intelligent problem solving. A fundamental element undelying tabu search is the use of flexible memory. From the standpoint of tabu search, flexible memory embodies the dual processes of creating and exploiting structures for taking advantage of history (hence combining the activities of acquaring and profiting from information).

The memory structures of tabu search operate by reference to four principal dimensions, consisting of recency, frequency, quality, and influence. These dimensions in turn are set against a background of logical structure and connectivity.



**Figure 1.** *Tabu Search method.*

**This page was last updated on 21-Nov-97**

|  |  |
| --- | --- |
| **X.** | **HYBRIDIZATION OF EC ALGORITHMS** |

**Contents of this chapter**

**X.1** GA + Branch and Bound

**X.1 GA + Branch and Bound**

[(Cotta, Aldana, Nebro, Troya, 1995)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#CO95)

GAs and B&B techniques are in some way antithetical methods for combinatorial-optimization-problem solving, at least with regard to the observable characteristics during the solving of such a problem. A straightforward comparison of both methods yields the following considerations:

B&B techniques allow a fast problem solving since the heuristics they incorporate ease a direct advance to solutions, rejecting those paths that do not drive the algorithm to an optimum (achieving it is guaranteed). In contrast, a GA is a quite slow procedure since it only takes in account the relative fitness of some approximate solutions. In fact, and unlike B&B techniques, those solutions that might be considered bad are not rejected because they may carry any important feature in order to solve the problem. It is important to realize that obtaining the optimum is not guaranteed by means of GAs.

GAs are more suitable for large instances of a problem. The key factor of their functioning is not the store space (whose utilization remains constant during the execution) but execution time which may be very large. As a matter of fact, indefinitely executing a GA will, theoretically, yield the optimal solution of the problem being solved. On the other hand, a B&B algorithm will fail solving a very large problem due to the resulting combinatorial explosion.

The features above make difficult to combine both techniques as shown next.

**X.1.1 *Direct collaboration. Issues***

A first possibility for GAs and B&B techniques to work together is executing both procedures independently and in parallel, that is, at the same level. Both processes will enclose the solution but coming from different directions: the GA will provide an upper bound of the optimal solution whilst the B&B algorithm produces a lower bound of it. Those bounds will become closer and closer along the execution. There are two ways of obtaining a benefit of this parallel execution:

Use the upper bounds provided by the GA to purge the problem queue of the B&B algorithm, deleting those subproblems whose lower bound is bigger than the one obtained from the GA.

Inject into the GA information about those subtours the B&B algorithm considers more promising. They would be the building blocks of the TSP representation.



**Figure 1**. *Direct collaboration scenario.*

These options are, however, difficult to carry out for many reasons:

Execution times are quite disparate, so establishing an effective information exchange is very difficult. For example, consider the negative effect resulting from injecting some high-valued building blocks (whose presence in the optimal solution is besides not guaranteed) in early stages of the GA, in which the population is a fairly homogeneous set of rather poor solutions. This might seriously affect diversity, thus polarizing further evolution in a wrong way.

There is a big gap between the accuracies of both methods. Specifically for the TSP, B&B algorithms start from an initial approximation that is usually within a 2% of the optimum, and work in that range seldom producing a subproblem with a lower bound much greater than the optimum. On the other hand, GAs do not usually achieve a high accuracy and when they do it occurs in late stages of evolution, so it is too late for a B&B algorithm to get a benefit by purging the problem queue.

Clearly, it is necessary to find an alternative approach in order to establish a productive collaboration between both techniques. At first and according to the functioning features mentioned, there exist two ways of harmonizing their performances, each one focusing in trying to make use of one of the collaboration options above. On the one hand, it is possible to reduce the combinatorial explosion which limits the effectiveness of a B&B algorithm by imposing some restrictions to the problem. On the other hand, parallel execution of many GAs may heighten their power.

The first option is represented by a strategy which has produced good results in a wide range of problems: build and Hybrid Genetic Algorithm (HGA). For the second one, it is more interesting a mixed approach (including restrictions in the B&B side too) than a pure one.

**X.1.2 *Branch and Bound working for a Genetic Algorithm***

In the model described next, the GA plays the role of a master and the B&B algorithm is incorporated as a tool of it. From the point of view of the GA, this is perhaps the most elegant option and the one that implies a bigger cohesion.

**X.1.2.1 Hybridizing a Genetic Algorithm**

As mentioned, the key feature of a GA is its robustness: binary representation and simple binary crossover and mutation operators provide problem-domain independence. However, such an algorithm will never be the best to solve a certain problem because, as in nature, every niche (optimization problem) is dominated by species (algorithms) specifically adapted to it. In fact, almost every specialized algorithm will outperform a GA.

It is possible, anyway, to improve a GA performance applying some techniques and knowledge used by other algorithms, although this implies a loss of robustness. There are three main principles in order to incorporate those items:

Use the encoding of the specialized algorithm.

Incorporate, where possible, those positive features of the specialized algorithm.

Adapt the genetic operators, using domain-specific heuristics and procedures.

Basically, the principles above show that a good use of all the theoretical basis of the specialized algorithm must be done. For example, the representation function of it will usually be based on experience in solving that problem, so its use will exploit that domain-knowledge. On the other hand, the specialized algorithm will normally use an efficient procedure to interpret that encoding and it can be also included in the HGA.

**X.1.2.2 A Hybrid Genetic Algorithm to solve the TSP**

To hybridize a GA incorporating B&B techniques and knowledge requires applying the principles mentioned above. A promising way of doing that is to build a hybrid operator with the philosophy of B&B. The idea underlying this option is shown next.



**Figure 2**. *Outline of the Hybrid Genetic Algorithm (HGA).*

As mentioned, the information related to the goodness of a certain tour is contained in the edges between nodes (in fact, they carry the weights of a connection). This information must be transmitted generation to generation using the genetic operators. Specifically, the information carried by some ancestors should be recombined to create some offsprings using a crossover operator. However, the classical existing operators usually introduce a great amount of new information since the produce offsprings including edges not appearing in any ancestor. That is why some advanced operators, such as Edge-Recombination, are created. This one guarantees that an offspring will inherit a 95% of its genetic information from its ancestors. The operator described next assures that an unique offspring will be created using only edges from the ancestors (i.e. without any implicit mutation).

The functioning of this operator is as follows: the B&B algorithm starts from a list of all edges in the graph with information related with associated weights. This operator is a restricted version of the B&B algorithm in which only the edges that appear in the ancestors are taken into account . The offspring obtained will not only 100% composed by its ancestors’ edges but it will also be the best solution that can be build using those edges (i.e. the operator has a high heuristic charge).

The net effect of this operator is to link low-cost subtours existing in the ancestors, or their construction. In fact, considering these elements as the building blocks of the problem the B&B algorithm would build them and glue together (injecting them into the GA as mentioned in 6.1).

**X.1.2.3 Aspects of the hybrid operator use**

It is clear that the restricted TSP solution carried out by the described operator is faster and does not have the memory limitations of the base case since it turns from solving a problem with (n^2)/2 edges to another with 4n edges in the worst case. However, this operator is going to be used a high number of times and its computational complexity is much higher than other operators such as edge-recombination, so the execution time of a GA that makes an intensive use of this operator might reach untenable values.

The first preventive measure in order to decrease the computational effort of the algorithm is a combined use of many crossover operators. Since the hybrid-operator complexity is determined by the number of edges taken into account, its use could be restricted to those occasions in which the ancestors share a good amount of edges (in the best case -the trivial one- only n edges are considered), using a standard operator in any other situation. A feasible model involves calculating the number of different edges of both ancestors, normalizing it (considering that the minimum is n and the maximum is 4n), and performing a random test based on this number in order to select which operator to use, the hybrid one or the standard one.

The described model has not only the advantage of reducing execution time but it also mitigate another drawback of the operator, the greediness of its heuristic functioning. This behaviour may even cause that no recombination takes place (when the best way to combine the edges is precisely one of the ancestors). Standard edge-recombination operator can produce offsprings whose fitness value is lower than the ancestors’ and introduce a little amount of mutation too. In this way, using the model above resembles a thermodynamic system: in early stages temperature (disparity between individuals) is high and the standard operator is often used, thus grouping building blocks. As the system freezes the heuristic operator becomes more used. In those late stages the probability of an important block not being carried due to the greediness of the operator is lower.

Another less desirable side effect of this operator is the tendency to a fast convergence that induces in the GA. Combining two rather poor individuals may frequently result in offsprings with a much higher fitness value than the ancestors. The apparition of these superindividuals [(Davis, 1991b)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#DAb91) may lead to a big loss of diversity and the sooner it occurs the more dangerous it is. This effect can be relieved introducing a higher mutation rate, with the risks that come with it.

**X.1.2.4 Improving the model. Parallelism**

In spite of the use of the described model, execution times (although decreased) are still high so another strategy must be used to optimize the process. One of the most suitable options is parallelizing it. There exists some alternatives in order to carry out that parallelization. The following ones can be mentioned:

Parallelizing the hybrid operator: this option would only reduce the execution time of the GA, remaining unaltered its behaviour.

Overlapping the hybrid operator with the rest of the GA: this option consists of the parallel launching of many hybrid crossover operations. The GA continues its functioning without waiting for these operations to be concluded. As they finish the generated offsprings are inserted into the population. This option do introduce qualitative behaviour differences in relation to the sequential version. The main one is the absolute disappearance of the generation concept since a great gapping occurs (individuals resulting from crossover operations launched many iterations before are inserted into the population).



**Figure 3**. *Outline of the Asynchronous Hybrid Genetic Algorithm (AHGA).*

It is possible that this overlapping provides a positive side effect: problems related to superindividuals where mentioned in 4.3.3. However, inserting them in the population many evolutionary cycles later may allow average fitness to increase enough not to suffer (or to suffer less) because of the presence of those individuals.

**X.1.3 *Branch and Bound and parallel evolution***

The other way to close up characteristics of B&B techniques and GAs is to heighten the power of the later ones in order to adequate them to a B&B speed. This heightening can be achieved by means of parallelly executing many GAs. This is a very general model since GAs can execute independently or establish some kind of collaboration between them, periodically interchanging a part of their populations [(Suárez, 1994)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#SU94).

**X.1.3.1 Describing the model**

In the simplest scenario, a B&B algorithm executes in parallel with a certain number of GAs. It is possible that many GAs provide good bounds in less time, although they will generally produce solution of similar magnitude and different structure, so the problem queue will seldom be purged (though in the worst case it equals the direct collaboration model). Consequently, it is necessary to develop a more complex model.



**Figure 4**. *Outline of the parallel evolution model.*

Notice one of the main problems of the B&B algorithm when applied to the TSP: a high number of edges must be considered. The complexity of the algorithm can be decreased by reducing this number of edges. Two extreme situations are the base case (in which every edge is taken into account) and the described hybrid operator (which only considers edges from two individuals). There exist intermediate positions between these extremes that consider a higher or lower number of edges. In order to delete those "bad" edges information from the GAs can be used.

The model outlined above can be described as follows. There is a generalized hybrid operator that recombines n individuals. Such operator is placed in a master process that controls m GAs, where n = k·m. Periodically, every GA send to the operator k individuals (e.g. the best k). The probable diversity in the structure of individuals provided by independent GAs will contribute to make that almost every edge suitable to be part of the optimal solution be included in any individual, and many non-suitable edges will not be taken into account.

**X.1.3.2 Variations and improvements**

There exist many variations that take as reference the strategy above. two fairly significant ones are:

Perform an information exchange between GAs as stated in any of the migratory models described in [(Suárez, 1994)](http://www.lcc.uma.es/~ccottap/semEC/apprefs/apprefs.html#SU94). Although this action may improve a GA, perhaps it would make diversity of solutions generated by independent GAs decrease, and this would not be good for the B&B stage.

Take n = k·m+1, that is, send to the hybrid operator not only the individuals provided by the GAs but also the best obtained so far. This is the B&B counterpart of GA elitism, since it guarantees that the best calculates solution is, at least, conserved at the expense of considering more edges.

Inject into the GAs the individual created by the hybrid operator. This is a dangerous action since it may cause convergence problems. In order to alleviate this drawback, some options are suggested:

* Do not send back the generated individual to the GAs until a certain number of generations is reached.
* Send back an individual to a GA only if the difference between its fitness and that of the best individual in this GA does not exceed a certain threshold.
* Send back to a GA the result of recombining the generated individual and the individual which that GA sent using a standard operator.

No experiences have been done in order to assure the goodness of the strategies above.

**This page was last updated on 21-Nov-97**