

# Introducción a la Computación Evolutiva

**Dr. Carlos A. Coello Coello**

**Departamento de Computación**

**CINVESTAV-IPN**

**Av. IPN No. 2508**

**Col. San Pedro Zacatenco**

**México, D.F. 07300**

**email: [ccoello@cs.cinvestav.mx](mailto:ccoello@cs.cinvestav.mx)**

**[http: //delta.cs.cinvestav.mx/~ccoello](http://delta.cs.cinvestav.mx/~ccoello)**

## Other Recent Approaches

There are two other recent approaches that we will also briefly discuss:

- A Simple Multimembered Evolution Strategy (SMES)
- The  $\alpha$  Constrained Method

## A Simple Multimembered Evolution Strategy

Mezura-Montes and Coello [2005] proposed an approach based on a  $(\mu + \lambda)$  evolution strategy. Individuals are compared using the following criteria (originally proposed by [Deb, 2000]):

1. Between two feasible solutions, the one with the highest fitness value wins.
2. If one solution is feasible and the other one is infeasible, the feasible solution wins.
3. If both solutions are infeasible, the one with the lowest sum of constraint violation is preferred.

## A Simple Multimembered Evolution Strategy

Additionally, the approach has 3 main mechanisms:

1. **Diversity Mechanism:** The infeasible solution which is closest to become feasible is retained in the population, so that it is recombined with feasible solutions.
2. **Combined Recombination:** Panmictic recombination is adopted, but with a combination of the discrete and intermediate recombination operators.
3. **Stepsize:** The initial stepsize of the evolution strategy is reduced so that finer movements in the search space are favored.

## A Simple Multimembered Evolution Strategy

This approach provided highly competitive results with respect to stochastic ranking, the homomorphous maps and ASCHEA while performing only 240,000 fitness function evaluations.

In more recent work, similar mechanisms were incorporated into a differential evolution algorithm, obtaining even better results.

The approach is easy to implement and robust.

## The $\alpha$ Constrained Method

This is a transformation method for constrained optimization introduced by Takahama [1999]. Its main idea is to define a satisfaction level for the constraints of a problem. The approach basically adopts a lexicographic order with relaxation of the constraints. Equality constraints can be easily handled through the relaxation of the constraints.

## The $\alpha$ Constrained Method

In [Takahama and Sakai, 2005], this approach is coupled to a modified version of Nelder and Mead's method. The authors argue that Nelder and Mead's method can be seen as an evolutionary algorithm in which, for example, the variation operators are: reflection, contraction and expansion. The authors also extend this method with a boundary mutation operator, the use of multiple simplexes, and a modification to the traditional operators of the method, as to avoid that the method gets easily trapped in a local optimum.

## The $\alpha$ Constrained Method

The approach was validated using a well-known benchmark of 13 test functions. Results were compared with respect to stochastic ranking. The number of evaluations performed was variable and ranged from 290,000 to 330,000 evaluations in most cases. The results found were very competitive, although the approach had certain sensitivity to the variation of some of its parameters.

## Special representations and operators

Some researchers have decided to develop special representation schemes to tackle a certain (particularly difficult) problem for which a generic representation scheme (e.g., the binary representation used in the traditional genetic algorithm) might not be appropriate.

## Special representations and operators

Due to the change of representation, it is necessary to design special genetic operators that work in a similar way than the traditional operators used with a binary representation. For example: Random Keys [Bean, 1992, 1994].

## Special representations and operators

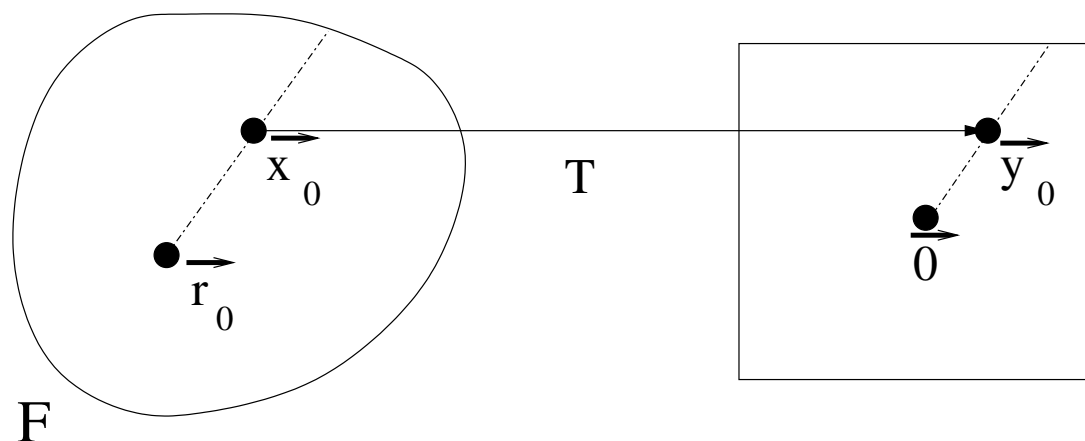
A more interesting type of approaches within this group are the so-called “Decoders”. The emphasis of these approaches is to map chromosomes from the infeasible region into the feasible region of the problem to solve. In some cases, special operators have also been designed in order to produce offspring that lie on the boundary between the feasible and the infeasible region.

## Special representations and operators

A more intriguing idea is to transform the whole feasible region into a different shape that is easier to explore. The most important approach designed along these lines are the “homomorphous maps” [Koziel & Michalewicz, 1999]. This approach performs a homomorphous mapping between an  $n$ -dimensional cube and a feasible search space (either convex or non-convex). The main idea of this approach is to transform the original problem into another (topologically equivalent) function that is easier to optimize by an evolutionary algorithm.

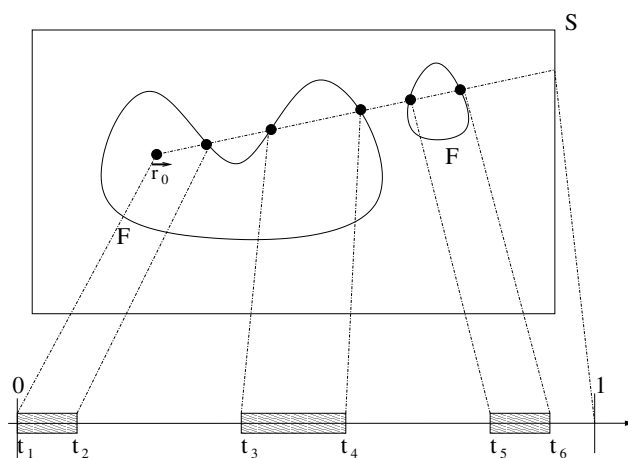
## Special representations and operators

The convex case is the following:



## Special representations and operators

The non-convex case is the following:



## Special representations and operators

The Homomorphous Maps (HM) was for some time, the most competitive constraint-handling approach available (until the publication of Stochastic Ranking). However, the implementation of the algorithm is more complex, and the experiments reported required a high number of fitness function evaluations (1,400,000).

## Special representations and operators

The version of HM for convex feasible regions is very efficient. However, the version for non-convex feasible regions requires a parameter  $v$  and a binary search procedure to find the intersection of a line with the boundary of the feasible region.

## Separation of constraints and objectives

Unlike penalty functions which combine the value of the objective function and the constraints of a problem to assign fitness, these approaches handle constraints and objectives separately. Some examples:

- **Coevolution:** Use two populations that interact with each other and have encounters [Paredis, 1994].

## Separation of constraints and objectives

- **Superiority of feasible points:** The idea is to assign always a higher fitness to feasible solutions [Powell & Skolnick, 1993; Deb, 2000].
- **Behavioral memory:** Schoenauer and Xanthakis [1993] proposed to satisfy, sequentially, the constraints of a problem.

## Separation of constraints and objectives

- **Use of multiobjective optimization concepts:** The main idea is to redefine the single-objective optimization of  $f(\vec{x})$  as a multiobjective optimization problem in which we will have  $m + 1$  objectives, where  $m$  is the total number of constraints. Then, any multi-objective evolutionary algorithm can be adopted [Coello et al., 2002; Deb, 2001]. Note however, that the use of multiobjective optimization is not straightforward, and several issues have to be taken into consideration.

## Separation of constraints and objectives

This last type of approach has been very popular in the last few years. For example:

- Surry & Radcliffe [1997] proposed COMOGA (Constrained Optimization by Multiobjective Optimization Genetic Algorithms) where individuals are Pareto-ranked based on the sum of constraint violation. Then, solutions can be selected using binary tournament selection based either on their rank or their objective function value.

## Separation of constraints and objectives

- Zhou et al. [2003] proposed a ranking procedure based on the Pareto Strength concept (introduced in SPEA) for the bi-objective problem, i.e. to count the number of individuals which are dominated for a given solution. Ties are solved by the sum of constraint violation (second objective in the problem). The Simplex crossover (SPX) operator is used to generate a set of offspring where the individual with the highest Pareto strength and also the solution with the lowest sum of constraint violation are both selected to take part in the population for the next generation.

## Separation of constraints and objectives

- Venkatraman and Yen [2005] proposed a generic framework divided in two phases: The first one treats the NLP as a constraint satisfaction problem i.e. the goal is to find at least one feasible solution. To achieve that, the population is ranked based only on the sum of constraint violation. The second phase starts when the first feasible solution was found. At this point, both objectives (original objective function and the sum of constraint violation) are taken into account and nondominated sorting [Deb, 2002] is used to rank the population.

## Separation of constraints and objectives

- Hernandez et al. [2004] proposed an approach named IS-PAES which is based on the Pareto Archive Evolution Strategy (PAES) originally proposed by Knowles and Corne [2000]. IS-PAES uses an external memory to store the best set of solutions found. Furthermore, it adopts a shrinking mechanism to reduce the search space. The multiobjective concept is used in this case as a secondary criterion (Pareto dominance is used only to decide whether or not a new solution is inserted in the external memory).

## Separation of constraints and objectives

### **Possible problems of the use of MO concepts:**

Runarsson and Yao [2005] presented a comparison of two versions of Pareto ranking in constraint space: (1) considering the objective function value in the ranking process and (2) without considering it. These versions were compared against a typical over-penalized penalty function approach. The authors found that the use of Pareto Ranking leads to bias-free search, then, they concluded that it causes the search to spend most of the time searching in the infeasible region; therefore, the approach is unable to find feasible solutions (or finds feasible solutions with a poor value of the objective function).

## Separation of constraints and objectives

### **Possible problems of the use of MO concepts:**

Note however, that “pure” Pareto ranking is rarely used as a mechanism to handle constraints, since some bias is normally introduced to the selection mechanism (when a constraint is satisfied, it makes no sense to keep using it in the Pareto dominance relationship).

## Hybrid methods

Within this category, we consider methods that are coupled with another technique (either another heuristic or a mathematical programming approach). Examples:

- Adeli and Cheng [1994] proposed a hybrid EA that integrates the penalty function method with the primal-dual method. This approach is based on sequential minimization of the Lagrangian method.

## Hybrid methods

- Kim and Myung [1997] proposed the use of an evolutionary optimization method combined with an augmented Lagrangian function that guarantees the generation of feasible solutions during the search process.
- **Constrained optimization by random evolution (CORE)**: This is an approach proposed by Belur [1997] which combines a random evolution search with Nelder and Mead's method [1965].

## Hybrid methods

- **Ant System (AS):** The main AS algorithm is a multi-agent system where low level interactions between single agents (i.e., artificial ants) result in a complex behavior of the whole ant colony. Although mainly used for combinatorial optimization, AS has also been successfully applied to numerical optimization [Bilchev & Parmee, 1995; Leguizamon, 2004]. Some of the recent research in this area focuses on the exploration of the boundary between the feasible and infeasible regions [Leguizamon & Coello, 2006].

## Hybrid methods

- **Simulated Annealing (SA):** Wah & Chen [2001] proposed a hybrid of SA and a genetic algorithm (GA). The first part of the search is guided by SA. After that, the best solution is refined using a GA. To deal with constraints, Wah & Chen use Lagrangian Multipliers.

## Hybrid methods

- **Artificial Immune System (AIS):** Hajela and Lee [1996] proposed a GA hybridized with an AIS (based on the negative selection approach). The idea is to adopt as antigens some feasible solutions and evolve (in an inner GA) the antibodies (i.e., the infeasible solutions) so that they are “similar” (at a genotypic level) to the antigens.

## Hybrid methods

- **Cultural Algorithms:** In this sort of approach, the main idea is to preserve beliefs that are socially accepted and discard (or prune) unacceptable beliefs. The acceptable beliefs can be seen as constraints that direct the population at the micro-evolutionary level. Therefore, constraints can influence directly the search process, leading to an efficient optimization process.

## Hybrid methods

- In other words, when using cultural algorithms, some sort of knowledge is extracted during the search process and is used to influence the evolutionary operators as to allow a more efficient search. The first versions of cultural algorithms for constrained optimization had some memory handling problems [Chung & Reynolds, 1996], but later on, they were improved using spatial data structures that allowed to handle problems with any number of decision variables [Coello & Landa, 2002].

## Test Functions

Michalewicz and Schoenauer [1996] proposed a set of test functions, which was later expanded by Runarsson and Yao [2000]. The current set contains 13 test functions. These test functions contain characteristics that are representative of what can be considered “difficult” global optimization problems for an evolutionary algorithm.

## Test Functions

Note however, that many other test functions exist. See for example:

Mezura Montes, Efrén and Coello Coello, Carlos A., **What Makes a Constrained Problem Difficult to Solve by an Evolutionary Algorithm**, Technical Report EVOCINV-01-2004, Evolutionary Computation Group at CINEVESTAV, Sección de Computación, Departamento de Ingeniería Eléctrica, CINEVESTAV-IPN, México, February 2004.

C. A. Floudas and P. M. Pardalos, **A Collection of Test Problems for Constrained Global Optimization Algorithms**, Number 455 in Lecture Notes in Computer Science. Springer-Verlag, 1990.

Christodoulos A. Floudas et al. (editors), **Handbook of Test Problems in Local and Global Optimization**, Kluwer Academic Publishers, Dordrecht, 1999.

## Test Functions (Current Benchmark)

Problem	n	Type of function	$\rho$	LI	NI	LE	NE
g01	13	quadratic	0,0003 %	9	0	0	0
g02	20	nonlinear	99,9973 %	1	1	0	0
g03	10	nonlinear	0,0026 %	0	0	0	1
g04	5	quadratic	27,0079 %	0	6	0	0
g05	4	nonlinear	0,0000 %	2	0	0	3
g06	2	nonlinear	0,0057 %	0	2	0	0
g07	10	quadratic	0,0000 %	3	5	0	0
g08	2	nonlinear	0,8581 %	0	2	0	0
g09	7	nonlinear	0,5199 %	0	4	0	0
g10	8	linear	0,0020 %	3	3	0	0
g11	2	quadratic	0,0973 %	0	0	0	1
g12	3	quadratic	4,7697 %	0	9 <sup>3</sup>	0	0
g13	5	nonlinear	0,0000 %	0	0	1	2

## Test Functions

Additional test functions have been recently proposed, and a new benchmark, consisting of 24 test functions (which include the 13 indicated in the previous slide) is now becoming more popular. See:

J. J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. N. Suganthan, C. A. Coello Coello, K. Deb, **Problem Definitions and Evaluation Criteria for the CEC 2006, Special Session on Constrained Real-Parameter Optimization**, Technical Report, Nanyang Technological University, Singapore, 2006.

## Test Case Generators

Michalewicz [2000] proposed a Test Case Generator for constrained parameter optimization techniques. This generator allows to build test problems by varying several features such as: dimensionality, multimodality, number of constraints, connectedness of the feasible region, size of the feasible region with respect to the whole search space and ruggedness of the objective function.

## Test Case Generators

The first version of this test problems generator had some problems because the functions produced were symmetric. This motivated the development of a new version called TCG-2 [Schmidt et al., 2000].

## Some Recommendations

- Study (and try first) traditional mathematical programming techniques (e.g., gradient-based methods, Nelder-Mead, Hooke-Jeeves, etc.).
- If interested in numerical optimization, try evolution strategies or differential evolution, instead of using genetic algorithms. Also, the combination of parents and offspring in the selection process tends to produce better performance.
- Pay attention to diversity. Keeping populations in which every individual is feasible is not always a good idea.
- Normalizing the constraints of the problem is normally a good idea.

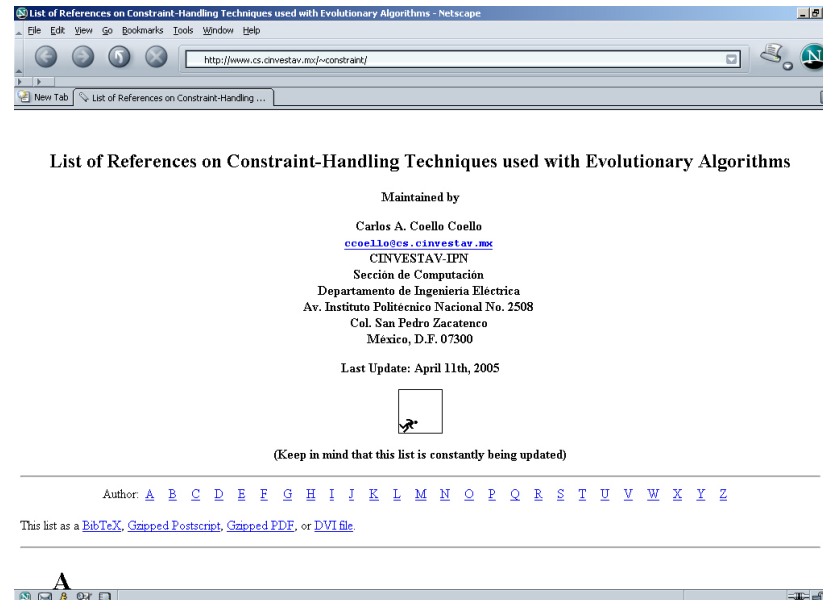
## Current Research Topics

- New constraint-handling approaches (e.g., based on multiobjective optimization concepts).
- Old constraint-handling techniques with new search engines (e.g., differential evolution, particle swarm optimization, ant colony, etc.).
- Hybrids of EAs with mathematical programming techniques (e.g., evolution strategy + simplex, use of Lagrange multipliers, etc.).

## Current Research Topics

- Test function generators (how to build them and make them reliable? Test functions available online?).
- New metrics that allow the evaluate the online performance of a constraint-handling technique.
- Special operators for exploring the boundary between the feasible and infeasible regions.

## To know more about constraint-handling techniques used with EAs



Please visit our constraint-handling repository located at:

**<http://www.cs.cinvestav.mx/~constraint>**

The repository currently (as of April 18th, 2007) had **524** bibliographic references.

## Suggested Readings

- Zbigniew Michalewicz and Marc Schoenauer, **Evolutionary Algorithms for Constrained Parameter Optimization Problems**, *Evolutionary Computation*, Vol. 4, No. 1, pp. 1–32, 1996.
- Carlos A. Coello Coello, **Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art**, *Computer Methods in Applied Mechanics and Engineering*, Vol. 191, No. 11–12, pp. 1245–1287, January 2002.

## Suggested Readings

- Thomas P. Runarsson and Xin Yao, **Stochastic Ranking for Constrained Evolutionary Optimization**, *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, September 2000.
- Alice E. Smith and David W. Coit, **Constraint Handling Techniques—Penalty Functions**, in Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter C 5.2. Oxford University Press and Institute of Physics Publishing, 1997.

## ¿Cómo Funcionan los Algoritmos Genéticos?

- Aunque los algoritmos genéticos son simples de describir y programar, su comportamiento puede ser muy complicado, y todavía existen muchas preguntas abiertas acerca de cómo funcionan y sobre el tipo de problemas en los que son más adecuados.

## ¿Cómo Funcionan los Algoritmos Genéticos?

- La teoría tradicional de los AGs (formulada originalmente por Holland en 1975) asume que, en un nivel muy general de descripción, los AGs trabajan descubriendo, enfatizando y recombinando buenos “bloques constructores” de soluciones en una manera altamente paralelizada.

## ¿Cómo Funcionan los Algoritmos Genéticos?

La idea básica aquí es que las buenas soluciones tienden a estar formadas de buenos bloques constructores (combinaciones de bits que confieren una aptitud alta a las cadenas en las que están presentes).

## ¿Cómo Funcionan los Algoritmos Genéticos?

- Holland (1975) introdujo la noción de esquemas para formalizar la noción de “bloques constructores”. Un esquema es un conjunto de cadenas de bits que pueden describirse mediante una plantilla hecha de unos, ceros y asteriscos que funcionan como comodines (pueden tomar cualquier valor dentro del alfabeto en uso).

## ¿Cómo Funcionan los Algoritmos Genéticos?

Por ejemplo, el esquema  $H = 1 * * * * 1$  representa el conjunto de todas las cadenas de 6 bits que empiezan y terminan con 1.

Usaremos la notación  $H$  para denotar “hiperplano” (Goldberg, 1989), porque los esquemas definen *hiperplanos* (“planos” de varias dimensiones en el espacio  $l$ -dimensional de las cadenas de  $l$  bits).

## ¿Cómo Funcionan los Algoritmos Genéticos?

Las cadenas que se ajustan a este esquema (por ejemplo, 100111 y 110011) son instancias de  $H$ . El esquema  $H$  tiene 2 bits definidos (posiciones que no son asteriscos), por lo que su orden es de 2. Su *longitud de definición* (la distancia entre sus bits definidos más extremos) es 5.

## ¿Cómo Funcionan los Algoritmos Genéticos?

- ¿Cómo procesa un AG los esquemas? Cualquier cadena de bits dada de longitud  $l$  es una instancia de  $2^l$  esquemas diferentes. Por ejemplo, la cadena 11 es una instancia de \*\* (todas las cadenas binarias posibles de 2 bits), \*1, 1\* y 11 (el esquema que contiene sólo la cadena 11).

## ¿Cómo Funcionan los Algoritmos Genéticos?

En otras palabras, cualquier población dada de  $n$  cadenas contiene instancias de entre  $2^l$  y  $N \times 2^l$  esquemas diferentes. Si todas las cadenas son idénticas, entonces hay  $2^l$  instancias de exactamente  $2^l$  esquemas diferentes; de lo contrario, el número es menor o igual a  $N \times 2^l$ .

## ¿Cómo Funcionan los Algoritmos Genéticos?

Esto significa que, en una cierta generación, mientras el AG está evaluando explícitamente las aptitudes de las  $n$  cadenas en la población, está también estimando implícitamente las aptitudes promedio de un número mucho mayor de esquemas, donde la aptitud promedio de un esquema se define como la aptitud promedio de todas las instancias posibles de ese esquema.

## ¿Cómo Funcionan los Algoritmos Genéticos?

Por ejemplo, en una población generada aleatoriamente de  $n$  cadenas, en promedio la mitad de las cadenas serán instancias de  $1***...*$  y la mitad serán instancias de  $0***...*$ . Las evaluaciones de las aproximadamente  $n/2$  cadenas que son instancias de  $1***...*$  nos da un estimado de la aptitud promedio de ese esquema (es un estimado porque las instancias evaluadas en poblaciones de tamaños típicos son sólo una pequeña muestra de todas las instancias posibles).

## ¿Cómo Funcionan los Algoritmos Genéticos?

Así como los esquemas no se representan o evalúan explícitamente por el AG, los estimados de las aptitudes promedio de los esquemas no se calculan o almacenan explícitamente por el AG. Sin embargo, el comportamiento del AG, en términos del incremento y decremento en números de instancias de esquemas dados en la población, puede describirse como si realmente estuviera calculando y almacenando estos promedios.

## ¿Cómo Funcionan los Algoritmos Genéticos?

- Podemos calcular la dinámica aproximada de este incremento y decremento en las instancias de los esquemas de la manera siguiente. Hagamos que  $H$  sea un esquema con al menos una instancia presente en la población en la generación  $t$ .

## ¿Cómo Funcionan los Algoritmos Genéticos?

Hagamos que  $m(H, t)$  sea el número de instancias de  $H$  en la generación  $t$ , y que  $\hat{u}(H, t)$  sea la aptitud promedio observada de  $H$  en la generación  $t$  (es decir, la aptitud promedio de las instancias de  $H$  en la población en la generación  $t$ ). Lo que queremos calcular es  $E(m(H, t + 1))$ , o sea el número esperado de instancias de  $H$  en la generación  $t + 1$ .

## ¿Cómo Funcionan los Algoritmos Genéticos?

Asumamos que usamos selección proporcional. Bajo este esquema, el número esperado de descendientes de una cadena  $x$  es igual a  $f(x)/\bar{f}(t)$ , donde  $f(x)$  es la aptitud de  $x$  y  $\bar{f}(t)$  es la aptitud promedio de la población en la generación  $t$ .

## ¿Cómo Funcionan los Algoritmos Genéticos?

Entonces, asumiendo que  $x$  está en la población en la generación  $t$ , y haciendo que  $x \in H$  denote “ $x$  es una instancia de  $H$ ”, e ignorando (por ahora) los efectos de la cruce y la mutación, tenemos:

$$E(m(H, t + 1)) = \sum_{x \in H} f(x) / \bar{f}(t) = (\hat{u}(H, t) / \bar{f}(t)) m(H, t) \quad (1)$$

## ¿Cómo Funcionan los Algoritmos Genéticos?

por definición, puesto que

$$\hat{u}(H, t) = \left( \sum_{x \in H} f(x) \right) / m(H, t)$$

para una  $x$  que se encuentre en la población en la generación  $t$ . De tal forma que aunque el AG no calcule explícitamente  $\hat{u}(H, t)$ , los incrementos o decrementos de las instancias de esquemas en la población dependen de esta cantidad.

## ¿Cómo Funcionan los Algoritmos Genéticos?

- Tanto la cruce como la mutación destruyen y crean instancias de  $H$ . Por ahora incluiremos sólo los efectos destructivos de la cruce y la mutación (aquellos que decrementan el número de instancias de  $H$ ). Si incluimos estos efectos, modificamos el lado derecho de la ecuación (1) para dar un límite inferior de  $E(m(H, t + 1))$ .

## ¿Cómo Funcionan los Algoritmos Genéticos?

Hagamos que  $p_c$  sea la probabilidad de cruza (de un punto) y supongamos que una instancia del esquema  $H$  se selecciona para ser padre. El esquema  $H$  se dice que “sobrevive” bajo la cruza de un punto si uno de sus hijos es también una instancia del esquema  $H$ .

## ¿Cómo Funcionan los Algoritmos Genéticos?

Podemos proporcionar un límite inferior de la probabilidad  $S_c(H)$  de que  $H$  sobrevivirá la cruce de un punto:

$$S_c(H) \geq 1 - p_c \left( \frac{\delta(H)}{l-1} \right)$$

donde  $\delta(H)$  es la longitud de definición de  $H$  y  $l$  es la longitud de las cadenas en el espacio de búsqueda.

## ¿Cómo Funcionan los Algoritmos Genéticos?

Esto es, las cruzas que ocurren dentro de la longitud de definición de  $H$  pueden destruir a  $H$  (es decir, pueden producir hijos que no son instancias de  $H$ ), así que multiplicamos la fracción de la cadena que  $H$  ocupa por la probabilidad de cruce para obtener un límite superior de la probabilidad de que el esquema será destruído.

## ¿Cómo Funcionan los Algoritmos Genéticos?

El valor es un límite superior porque algunas cruza dentro de las posiciones definidas de un esquema no lo destruirán (por ejemplo, si dos cadenas idénticas se cruzan). Al sustraer este valor de 1 obtenemos un límite inferior. En resumen, la probabilidad de supervivencia bajo cruza es alta para esquemas más cortos.

## ¿Cómo Funcionan los Algoritmos Genéticos?

Ahora cuantificaremos los efectos de perturbación de la mutación. Hagamos que  $p_m$  sea la probabilidad de mutación y que  $S_m(H)$  sea la probabilidad de que el esquema  $H$  sobrevivirá bajo la mutación de una instancia de  $H$ . Esta probabilidad es igual a:

$$(1 - p_m)^{o(H)}$$

donde  $o(H)$  es el orden de  $H$  (es decir, el número de bits definidos en  $H$ ).

## ¿Cómo Funcionan los Algoritmos Genéticos?

Esto es, para cada bit, la probabilidad de que el bit no se mutará es  $1 - p_m$ , de manera que la probabilidad de que bits no definidos del esquema  $H$  se muten es esta cantidad multiplicada por sí misma  $o(H)$  veces. En resumen, la probabilidad de supervivencia bajo mutación es más alta para esquemas de menor orden.

## ¿Cómo Funcionan los Algoritmos Genéticos?

Estos efectos de perturbación pueden usarse para modificar la ecuación (1):

$$E(m(H, t + 1)) \geq \frac{\hat{u}(H, t)}{\vec{f}(t)} m(H, t) \left( 1 - p_c \frac{\delta(H)}{l-1} \right) [(1 - p_m)^{o(H)}]$$

A esta expresión se le conoce como el **Teorema de los Esquemas** (Holland, 1975), y describe el crecimiento de un esquema de una generación a la siguiente.

## ¿Cómo Funcionan los Algoritmos Genéticos?

El **Teorema de los Esquemas** frecuentemente se interpreta de tal forma que implica que los esquemas cortos y de bajo orden cuya aptitud promedio se mantiene por encima de la media, recibirán un número de muestras que crece exponencialmente sobre el tiempo.

## ¿Cómo Funcionan los Algoritmos Genéticos?

- La razón por la que se dice que los esquemas cortos y de bajo orden reciben un número de muestras que se incrementa exponencialmente con el tiempo es porque el número de muestras de esos esquemas que no son perturbados y permanecen sobre la aptitud promedio se incrementan en un factor de  $\hat{u}(H, t) / \bar{f}(t)$  a cada generación.

## ¿Cómo Funcionan los Algoritmos Genéticos?

- El **Teorema de los Esquemas** es un límite inferior puesto que lidia solamente con los efectos destructivos de la cruza y la mutación. Sin embargo, se cree que la cruza es la fuente de mayor poder del AG, pues tiene la capacidad de recombinar las instancias de esquemas favorables para formar otros igualmente buenos o mejores.

## ¿Cómo Funcionan los Algoritmos Genéticos?

- Esta suposición de que los AGs trabajan de esta manera se conoce como la **Hipótesis de los Bloques Constructores** (Goldberg, 1989).

## ¿Cómo Funcionan los Algoritmos Genéticos?

- El efecto de la selección es sesgar gradualmente el procedimiento de muestreo hacia instancias de esquemas cuya aptitud se estime estén sobre el promedio. Con el paso del tiempo, el estimado de la aptitud promedio de un esquema debiera, en principio, volverse cada vez más preciso puesto que el AG está muestreando más y más instancias de este esquema.

## ¿Cómo Funcionan los Algoritmos Genéticos?

- El Teorema de los Esquemas y la Hipótesis de los Bloques Constructores lidian primordialmente con el papel de la selección y la cruce en los AGs, pero ¿cuál es el papel de la mutación? Holland (1975) propuso que la mutación previene la pérdida de diversidad en una posición cualquiera. Es una especie de “póliza de seguro” contra fijaciones en una cadena cromosómica.

## Críticas al Teorema de los Esquemas

- El Teorema de los Esquemas es realmente una desigualdad “débil”, no un “teorema”.
- Las siguientes afirmaciones sobre el teorema de los esquemas no son del todo demostrables:
  - a) Los esquemas por arriba del promedio se incrementan exponencialmente con el tiempo.

## Críticas al Teorema de los Esquemas

- b) Los esquemas por arriba del promedio se exploran rápidamente en paralelo sin alentar de manera significativa la búsqueda.
- c) Aproximadamente se procesan  $n^3$  esquemas de manera útil y en paralelo por cada generación