La Importancia de la Representación en los Algoritmos Genéticos (Parte I)

Carlos A. Coello Coello
LANIA, A.C.
Rébsamen 80, Apartado Postal 696
Xalapa, Veracruz 91090, México
ccoello@xalapa.lania.mx

Resumen

Aunque la representación binaria suele considerarse universal en los algoritmos genéticos, existen varios dominios donde ésta se ve severamente limitada, haciéndose necesario utilizar esquemas de codificación alternativos. Esta primera parte analiza diferentes esquemas alternativos de representación que van desde los códigos de Gray hasta los algoritmos genéticos desordenados, describiendo en cada caso su motivación y algunas de sus aplicaciones reportadas en la literatura especializada.

1. Introducción

Las capacidades para procesamiento de datos de las técnicas de computación evolutiva dentro de una amplia gama de dominios han sido reconocidas en los últimos años y han recibido mucha atención por parte de científicos que trabajan en diversas disciplinas. Dentro de estas técnicas evolutivas, quizás la más popular sea el algoritmo genético (AG) [1,2].

Siendo una técnica heurística estocástica, el algoritmo genético no necesita información específica para guiar la búsqueda. Su estructura presenta analogías con la teoría biológica de la evolución, y se basa en el principio de la supervivencia del más apto [3]. Por lo tanto, el AG puede verse como una "caja negra" que puede conectarse a cualquier aplicación en particular.

En general, se necesitan los cinco componentes básicos siguientes para implementar un AG que resuelva un problema cualquiera [4]:

1. Una representación de soluciones potenciales al problema.

2. Una forma de crear una población inicial de soluciones potenciales (esto se efectúa normalmente de manera aleatoria, pero también pueden usarse métodos determinísticos).

3. Una función de evaluación que juega el papel del ambiente, calificando a las soluciones producidas en términos de su "aptitud".

4. Operadores genéticos que alteran la composición de los descendientes (normalmente se usan la cruza y la mutación).

5. Valores para los diversos parámetros utilizados por el algoritmo genético (tamaño de la población, probabilidad de cruza y mutación, número máximo de generaciones, etc.)

1	0	0	1	1	0	1

Figura 1 : Un ejemplo de una cadena binaria.

En este artículo hablaremos exclusivamente del primero de estos componentes: la representación usada por el algoritmo genético.

La representación tradicional usada para codificar un conjunto de soluciones es el esquema binario en el cual un cromosoma[1] es una cadena de la forma $\langle b_1, b_2, \ldots, b_m \rangle$ (ver figura 1), donde $b_1, b_2, \ldots, b_m$ se denominan *alelos* (ya sea ceros o unos).

Hay varias razones por las cuales suele usarse la codificación binaria en los AGs, aunque la mayoría de ellas se remontan al trabajo pionero de Holland en el área. En su libro, Holland [3] dio una justificación teórica para usar codificaciones binarias. Holland comparó dos representaciones diferentes que tuvieran aproximadamente la misma capacidad de acarreo de información, pero de entre ellas, una tenía pocos alelos y cadenas largas (por ejemplo, cadenas binarias de 80 bits de longitud) y la otra tenía un número elevado de alelos y cadenas cortas (por ejemplo, cadenas decimales de longitud 24). Nóteses que 2^{80} (codificación binaria) $\approx 10^{24}$ (codificación decimal). Holland [3] argumentó que la primera codificación da pie a un grado más elevado de 'paralelismo implícito' porque permite más "esquemas" que la segunda (11^{24} contra 3^{80}). El número de esquemas de una cadena se calcula usando $(c+1)^l$, donde c es la cardinalidad del alfabeto y l es la longitud de la cadena[2]. Un "esquema" es una plantilla que describe un subconjunto de cadenas que comparten ciertas similitudes en algunas posiciones a lo largo de su longitud [1,3].

El hecho de contar con más esquemas favorece la diversidad e incrementa la probabilidad de que se formen buenos "bloques constructores" (es decir, la porción de un cromosoma que le produce una aptitud elevada a la cadena en la cual está presente) en cada generación, lo que en consecuencia mejora el desempeño del AG con el paso del tiempo de acuerdo al teorema de los esquemas [1,3]. El 'paralelismo implícito' de los AGs, demostrado por Holland [3], se refiere al hecho de que mientras el AG calcula las aptitudes de los individuos en una población, estima de forma implícita las aptitudes promedio de un número mucho más alto de cadenas cromosómicas a través del cálculo de las aptitudes promedio observadas en los "bloques constructores" que se detectan en la población.

---

[1] Un **cromosoma** es una estructura de datos que contiene una 'cadena' de parámetros de diseño o genes.

[2] La razón por la que se suma uno a la cardinalidad es porque en los esquemas se usa un símbolo adicional (normalmente el asterisco o el símbolo de número) para indicar que "no nos importa" el valor de esa posición.

Por lo tanto, de acuerdo a Holland [3], es preferible tener muchos genes[3] con pocos alelos posibles que contar con pocos genes con muchos alelos posibles. Esto es sugerido no sólo por razones teóricas (de acuerdo al teorema de los esquemas formulado por Holland), sino que también tiene una justificación biológica, ya que en genética es más usual tener cromosomas con muchas posiciones y pocos alelos por posición que pocas posiciones y muchos alelos por posición [3].
Sin embargo, Holland [3] también demostró que el paralelismo implícito de los AGs no impide usar alfabetos de mayor cardinalidad [3], aunque debe estarse siempre consciente de que el alfabeto binario es el que ofrece el mayor número de esquemas posibles por bit de información si se compara con cualquier otra codificación posible [1,4]. No obstante, ha habido un largo debate en torno a cuestiones relacionadas con estos alfabetos no binarios, principalmente por parte de los especialistas en aplicaciones de los AGs.
Como veremos en este artículo, el uso de la representación binaria tiene varias desventajas cuando el AG se usa para resolver ciertos problemas del mundo real. Por ejemplo, si tratamos de optimizar una función con alta dimensionalidad (digamos, con 50 variables), y queremos trabajar con una buena precisión (por ejemplo, cinco decimales), entonces el mapeo de números reales a binarios generará cadenas extremadamente largas (del orden de 1000 bits en este caso), y el AG tendrá muchos problemas para producir resultados aceptables en la mayor parte de los casos, a menos que usemos procedimientos y operadores especialmente diseñados para el problema en cuestión.
Ronald [5] resume las principales razones por las que una codificación binaria puede no resultar adecuada en un problema dado:

- Epístasis : el valor de un bit puede suprimir las contribuciones de aptitud de otros bits en el genotipo[4].

- Representación natural : algunos problemas (como el del viajero) se prestan de manera natural para la utilización de representaciones de mayor cardinalidad que la binaria (por ejemplo, el problema del viajero se presta de manera natural para el uso de permutaciones de enteros decimales).

- Soluciones ilegales : los operadores genéticos utilizados pueden producir con frecuencia (e incluso todo el tiempo) soluciones ilegales si se usa una representación binaria.

---

[3] Se denomina **gene** o **gen** a cualquier posición a lo largo de una cadena que representa a un individuo.

[4] El **genotipo** es la cadena cromosómica utilizada para almacenar la información contenida en un individuo, mientras que el **fenotipo** se refiere a los valores que toman las variables tras "decodificar" el contenido cromosómico de un individuo.

En el resto de este artículo discutiremos algunos esquemas de representación alternativos que han sido propuestos recientemente para lidiar con éstas y otras limitaciones de la representación binaria.

2. Códigos de Gray

Un problema que fue notado desde los inicios de la investigación en AGs fue que el uso de la representación binaria no mapea adecuadamente el espacio de búsqueda con el espacio de representación [6]. Por ejemplo, si codificamos en binario los enteros 5 y 6, los cuales están adyacentes en el espacio de búsqueda, sus equivalentes en binario serán el 101 y el 110, los cuales difieren en 2 bits (el primero y el segundo de derecha a izquierda) en el espacio de representación.

A este fenómeno se le conoce como el *risco de Hamming* (*Hamming cliff*) [7], y ha conducido a los investigadores a proponer una representación alternativa en la que la propiedad de adyacencia existente en el espacio de búsqueda pueda preservarse en el espacio de representación. La codificación de Gray es parte de una familia de representaciones que caen dentro de esta categoría [8].

Podemos convertir cualquier número binario a un código de Gray haciendo XOR a sus bits consecutivos de derecha a izquierda. Por ejemplo, dado el número 0101 en binario, haríamos[5]: $1 \oplus 0 = 1$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, produciéndose (el último bit de la izquierda permanece igual) 0111, el cual es el código de Gray equivalente.

Algunos investigadores han demostrado empíricamente que el uso de códigos de Gray mejora el desempeño del AG al aplicarse a las funciones de prueba clásicas de De Jong [9] (ver por ejemplo [7, 10]). De hecho, Mathias y Whitley [11] encontraron que la codificación de Gray no sólo elimina los riscos de Hamming, sino que también altera el número de óptimos locales en el espacio de búsqueda así como el tamaño de las buenas regiones de búsqueda (aquellas que nos conducirán a la vecindad del óptimo global). En su trabajo, Mathias y Whitley mostraron empíricamente que un mutador aleatorio del tipo "escalando la colina" es capaz de encontrar el óptimo global de la mayor parte de las funciones de prueba utilizadas cuando se emplea la codificación de Gray, a pesar de que algunas de ellas fueron diseñadas explícitamente para presentar dificultades a los algoritmos de búsqueda tradicionales (sean evolutivos o no).

3. Codificando Números Reales

Aunque los códigos de Gray pueden ser muy útiles para representar enteros, el problema de mapear correctamente el espacio de búsqueda en el espacio de representación se vuelve más serio cuando tratamos de codificar números reales. En el enfoque tradicional [12], se usa un número binario para representar un número real, definiendo límites inferiores y superiores para cada variable, así como la precisión deseada. Por ejemplo, si queremos codificar una variable que va de 0.35 a 1.40 usando una precisión de 2 decimales, necesitaríamos $\log_2$ (140-

---

[5] $\oplus$ indica XOR.

35) ≈ 7 bits para representar cualquier número real dentro de ese rango. Sin embargo, en este caso, tenemos el mismo problema del que hablamos anteriormente, porque el número 0.38 se representaría como 0000011, mientras que 0.39 se representaría como 0000101.

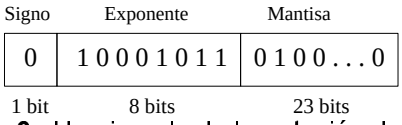

Figura 2 : Un ejemplo de la notación del IEEE

Aunque se usen códigos de Gray, existe otro problema más importante cuando tratamos de desarrollar aplicaciones del mundo real: la alta dimensionalidad. Si tenemos demasiadas variables, y queremos una muy buena precisión para cada una de ellas, entonces las cadenas binarias que se produzcan se volverán extremadamente largas, y el AG tenderá a tener un desempeño pobre. Si en vez de usar este tipo de mapeo adoptamos algún formato binario estándar para representar números reales, como por ejemplo el estándar del IEEE para precisión simple, en el cual un número real se representa usando 32 bits, de los cuales 8 se usan para el exponente usando una notación en exceso-127, y la mantisa se representa con 23 bits (ver figura 2 [13]), podríamos manejar un rango relativamente grande de números reales usando una cantidad fija de bits (por ejemplo, de 2^{-126} a 2^{127} si usamos el estándar de precisión simple antes descrito). Sin embargo, el proceso de decodificación sería más costoso (computacionalmente hablando) y el mapeo entre el espacio de representación y el de búsqueda sería mucho más complejo que cuando se usa una representación binaria simple, porque cualquier pequeño cambio en el exponente produciría grandes saltos en el espacio de búsqueda, mientras que perturbaciones en la mantisa podrían no cambiar de manera significativa el valor numérico codificado.

Mientras los teóricos afirman que los alfabetos pequeños son más efectivos que los alfabetos grandes, los prácticos han mostrado a través de una cantidad significativa de aplicaciones del mundo real (particularmente problemas de optimización numérica) que el uso directo de números reales en un cromosoma funciona mejor en la práctica que la representación binaria tradicional [14, 15].

El uso de números reales en una cadena cromosómica (ver figura 3) ha sido común en otras técnicas de computación evolutiva tales como las estrategias evolutivas [16] y la programación evolutiva [17], donde la mutación es el operador principal.

Sin embargo, los teóricos de los AGs han criticado fuertemente el uso de valores reales en los genes de un cromosoma, principalmente porque esta representación de cardinalidad más alta tiende a hacer que el comportamiento del AG sea más errático y difícil de predecir. Debido a esto, se han diseñado varios operadores especiales en los años recientes, para emular el efecto de la cruza y la mutación en los alfabetos binarios [12, 15, 18].

2.15	1.89	0.43	3.14	0.27	7.93	5.11

Figura 3 : Un ejemplo de un algoritmo genético con representación real.

Los prácticos argumentan que una de las principales capacidades de los AGs que usan representación real es la de explotar la "gradualidad" de las funciones de variables continuas[6].
Esto significa que los AGs con codificación real pueden lidiar adecuadamente con los "riscos" producidos cuando las variables utilizadas son números reales, porque un cambio pequeño en la representación es mapeado como un cambio pequeño en el espacio de búsqueda [12, 15].
En un intento por reducir la brecha entre la teoría y la práctica, algunos investigadores han desarrollado un marco teórico que justifique el uso de alfabetos de más alta cardinalidad [12, 15, 19, 20], pero han habido pocos consensos en torno a los problemas principales, por lo que el uso de AGs con codificación real sigue siendo una elección que se deja al usuario.

1	4	5	6	7	9

Figura 4 : Una representación entera de números reales. La cadena completa es decodificada como un solo número real multiplicando y dividiendo cada dígito de acuerdo a su posición.

Se han usado también otras representaciones de los números reales. Por ejemplo, el uso de enteros para representar cada dígito ha sido aplicado exitosamente a varios problemas de optimización [21,22]. La figura 4 muestra un ejemplo en el cual se representa el número 1.45679 usando enteros. En este caso, se supone una posición fija para el punto decimal en cada variable, aunque esta posición no tiene que ser necesariamente la misma para el resto de las variables codificadas en la misma cadena. La precisión está limitada por la longitud de la cadena, y puede incrementarse o decrementarse según se desee. Los operadores de cruza tradicionales (un punto, dos puntos y uniforme) pueden usarse directamente en esta representación, y la mutación puede consistir en generar un dígito aleatorio para una cierta posición o bien en producir una pequeña perturbación (por ejemplo ± 1) para evitar saltos extremadamente grandes en el espacio de búsqueda. Esta representación pretende ser un compromiso entre un AG con codificación real y una representación binaria de números reales, manteniendo lo mejor de ambos esquemas al incrementar la cardinalidad del alfabeto utilizado, pero manteniendo el uso de los operadores genéticos tradicionales casi sin cambios.

---

[6] "Gradualidad" se refiere a los casos en los cuales un cambio pequeño en las variables se traduce en un cambio pequeño en la función.

145679	67893	37568	95432

Figura 5 : Otra representación entera de números reales. En este caso, cada gene contiene un número real representado como un entero largo.

Alternativamente, podríamos también usar enteros largos para representar números reales (ver figura 5), pero los operadores tendrían que redefinirse de la misma manera que al usar números reales. El uso de este esquema de representación como una alternativa a los AGs con codificación real parece, sin embargo, un tanto improbable, ya que se tendrían que hacer sacrificios notables en la representación, y los únicos ahorros importantes que se lograrían serían en términos de memoria (el almacenamiento de enteros toma menos memoria que el de números reales). No obstante, este esquema ha sido usado en algunas aplicaciones del mundo real [14].

4. Representaciones de Longitud Variable

En algunos problemas el uso de alfabetos de alta cardinalidad puede no ser suficiente, pues además puede requerirse el empleo de cromosomas de longitud variable para lidiar con cambios que ocurran en el ambiente con respecto al tiempo (por ejemplo, el decremento/incremento de la precisión de una variable o la adición/remoción de variables).

Algunas veces, puede ser posible introducir símbolos en el alfabeto que sean considerados como posiciones "vacías" a lo largo de la cadena, con lo que se permite la definición de cadenas de longitud variable aunque los cromosomas tengan una longitud fija. Ese es, por ejemplo, el enfoque utilizado en [23] para diseñar circuitos eléctricos combinatorios. En ese caso, el uso de un símbolo llamado WIRE, el cual representa la ausencia de compuerta, permitió cambiar la longitud de la expresión Booleana generada a partir de una matriz bi-dimensional.

Sin embargo, en otros dominios, este tipo de simplificación puede no ser posible y deben idearse representaciones alternativas. Por ejemplo, en problemas que tienen decepción parcial o total [24] (es decir, en aquellos problemas en los que los bloques constructores de bajo orden no guían al AG hacia el óptimo y no se combinan para formar bloques constructores de orden mayor), un AG no tendrá un buen desempeño sin importar cuál sea el valor de sus parámetros (tamaño de población, porcentajes de cruza y mutación, etc.).

Para lidiar con este tipo de problemas en particular, Goldberg et al. [25,26,27] propusieron el uso de un tipo especial de AG de longitud variable el cual usa poblaciones de tamaño variable. A este AG especial se le denominó 'desordenado' (*messy GA* o mGA) en contraposición con el AG estándar (u ordenado), que tiene longitud y tamaño de población fijos [28].

La idea básica de los AGs desordenados es empezar con cromosomas cortos, identificar un conjunto de buenos bloques constructores y después incrementar la

longitud del cromosoma para propagar estos buenos bloques constructores a lo largo del resto de la cadena.

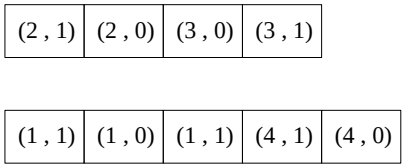

Figura 6 : Dos ejemplos de cadenas válidas en un algoritmo genético desordenado

La representación usada por los AGs desordenados es muy peculiar, puesto que cada bit está realmente asociado con una posición en particular a lo largo de la cadena, y algunas posiciones podrían ser asignadas a más de un bit (a esto se le llama *sobre-especificación*) mientras que otras podrían no ser asignadas a ninguno (a esto se le llama *sub-especificación*). Consideremos, por ejemplo, a las dos cadenas mostradas en la figura 6, las cuales constituyen cromosomas válidos para un AG desordenado (estamos suponiendo cromosomas de 4 bits).

La notación adoptada en este ejemplo usa paréntesis para identificar a cada gene, el cual se define como un par consistente de su posición a lo largo de la cadena (el primer valor) y el valor del bit en esa posición (estamos suponiendo un alfabeto binario). En el primer caso, la primera y la cuarta posición no están especificadas, y la segunda y la tercera están especificadas dos veces.

Para lidiar con la sobre-especificación pueden definirse algunas reglas determinísticas muy sencillas. Por ejemplo, podemos usar sólo la primera definición de izquierda a derecha para una cierta posición. Sin embargo, para la sub-especificación tenemos que hacer algo más complicado, porque una cadena sub-especificada realmente representa a un "esquema candidato" en vez de un cromosoma completo. Por ejemplo, la primera cadena de las antes descritas representa al esquema *10* (el * significa "no me importa"). Para calcular la aptitud de una cadena sub-especificada, podemos usar un explorador local del tipo "escalando la colina" que nos permita localizar el óptimo local y la información obtenida la podemos utilizar para reemplazar a los "no me importa" del esquema. A esta técnica se le denomina "plantillas competitivas" [26].

Los AGs desordenados operan en 2 fases [26]: la "fase primordial" y la "fase yuxtaposicional". En la primera, se generan esquemas cortos que sirven como los bloques constructores en la fase yuxtaposicional en la cual éstos se combinan. El problema es cómo decidir qué tan largos deben ser estos esquemas "cortos". Si son demasiado cortos, pueden no contener suficiente material genético como para resolver el problema deseado; si son demasiado largos, la técnica puede volverse impráctica debido a la "maldición de la dimensionalidad" (tendríamos que generar y evaluar demasiados cromosomas).

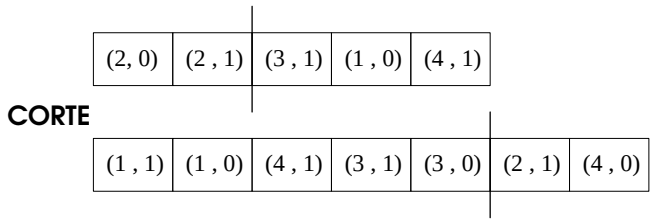

Figura 7 : Un ejemplo del operador de "corte" en un AG desordenado. La línea gruesa indica el punto de corte.

Durante la fase primordial generamos entonces estos esquemas cortos y evaluamos sus aptitudes. Después de eso, aplicamos sólo selección a la población (sin cruza o mutación) para propagar los buenos bloques constructores, y se borra la mitad de la población a intervalos regulares [28].
Después de un cierto número (predefinido) de generaciones, terminamos la fase primordial y entramos a la fase yuxtaposicional. A partir de este punto, el tamaño de la población permanecerá fijo, y usaremos selección y dos operadores especiales llamados "corte" y "unión" [25]. El operador de corte simplemente remueve una porción del cromosoma, mientras que el de unión junta dos segmentos cromosómicos. Considere los ejemplos mostrados en las figuras 7 y 8.

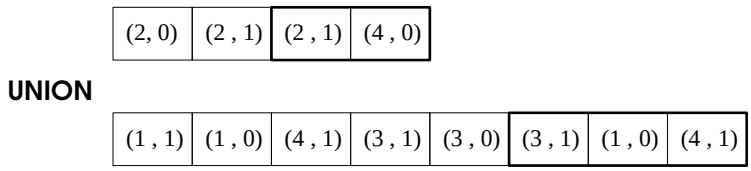

Figura 8 : Un ejemplo del operador "unión" en un AG desordenado. La línea gruesa muestra la parte de la cadena que fue agregada.

Debido a la naturaleza del AG desordenado, las cadenas producidas por los operadores de corte y unión siempre serán válidas. Si los bloques constructores producidos en la fase primordial acarrean suficiente información, entonces el AG desordenado será capaz de arribar al óptimo global aunque el problema tenga decepción [27].
Aunque es sin duda muy prometedor, los inconvenientes prácticos del AG desordenado [28] han impedido su uso extendido, y actualmente se reportan relativamente pocas aplicaciones (ver por ejemplo [29, 30, 31]).

Referencias Bibliográficas

[1] Goldberg, D. E. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, Massachusetts : Addison-Wesley Publishing Co.

[3] Coello Coello, C. A. (1995), Introducción a los Algoritmos Genéticos, *Soluciones Avanzadas. Tecnologías de Información y Estrategias de Negocios*, Año 3, Número 17, pp. 5-11.

[3] Holland, J. H. (1975), *Adaptation in Natural and Artificial Systems*, Ann Harbor : University of Michigan Press.

[4] Michalewicz, Z. (1992), *Genetic Algorithms + Data Structures = Evolution Programs*, second edition, Springer-Verlag.

[5] Ronald, S. (1997) Robust Encodings in Genetic Algorithms, *en* D. Dasgupta & Z. Michalewicz, editors, Evolutionary Algorithms in Engineering Applications, Springer-Verlag, pp. 30-44.

[6] Hollstien, R. B. (1971), *Artificial Genetic Adaptation in computer control systems*, PhD thesis, University of Michigan, Ann Harbor, Michigan.

[7] Caruana, R. & Schaffer, J. D. (1988), Representation and Hidden Bias: Gray vs. Binary Coding for Genetic Algorithms, *en* 'Proceedings of the Fifth International Conference on Machine Learning', Morgan Kauffman Publishers, San Mateo, California, pp.132-161.

[8] Whitley, D., Rana, S. & Heckendorn, R. (1998), Representation Issues in Neighborhood Search and Evolutionary Algorithms, *en* D. Quagliarella, J. Périaux, C. Poloni & G. Winter, eds, 'Genetic Algorithms and Evolution Strategies in Engineering and Computer Science. Recent Advances and Industrial Applications', John Wiley and Sons, West Sussex, England, chapter 3, pp.39-57.

[9] De Jong, K. (1975), *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, PhD thesis, University of Michigan.

[10] Mathias, K. E. & Whitley, L. D. (1994*a*), Transforming the search space with Gray coding, *en* J.D. Schaffer, ed., 'Proceedings of the IEEE International Conference on Evolutionary Computation', IEEE Service Center, Piscataway, New Jersey, pp.513-518.

[11] Mathias, K. E. & Whitley, L. D. (1994*b*), 'Changing Representations During Search: A Comparative Study of Delta Coding', *Evolutionary Computation* **2**(3),249-278.

[12] Wright, A. H. (1991), Genetic algorithms for real parameter optimization, *en* G. J. E. Rawlins, editor, 'Foundations of Genetic Algorithms', Morgan Kaufmann Publishers, San Mateo, California, pp.205-218.

[13] Scragg, G. W. (1992), *Computer Organization. A Top-Down Approach*, McGraw-Hill, New York.

[14] Davis, L., editor (1991), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.

[15] Eshelman, L. J. & Schaffer, J. D. (1993), Real-coded genetic algorithms and interval-schemata, *en* L.D. Whitley, ed., 'Foundations of Genetic Algorithms 2', Morgan Kaufmann Publishers, San Mateo, California, pp.187-202.

[16] Schwefel, H. P. (1981), *Numerical Optimization of Computer Models*, John Wiley and sons, Great Britain.

[17] Fogel, D. B. & Stayton, L. C. (1994), 'On the Effectiveness of Crossover in Simulated Evolutionary Optimization', *BioSystems* **32**, pp. 171-182.

[18] Deb, K. & Agrawal, R. B. (1995), 'Simulated Binary Crossover for Continuous Search Space', *Complex Systems* **9**, pp. 115-148.

[19] Goldberg, D. E. (1990), Real-coded genetic algorithms, virtual alphabets and blocking, Technical Report 90001, University of Illinois at Urbana-Champaign, Urbana, Illinois.

[20] Surry, P. D. & Radcliffe, N. J. (1997), Real representations, *en* R.K. Belew & M.D. Vose, editors, 'Foundations of Genetic Algorithms 4', Morgan Kaufmann, San Mateo, California, pp. 343-363.

[21] Coello Coello, C. A., Hernández, F. S. & Farrera, F. A. (1997), 'Optimal design of reinforced concrete beams using genetic algorithms', *Expert Systems with Applications : An International Journal* **12**(1), pp. 101-108.

[22] Coello Coello, C. A., Christiansen, A. D. & Aguirre, A. H. (1998) , 'Using a new GA-based multiobjective optimization technique for the design of robot arms', *Robotica*, Vol. 16, pp. 401-414.

[23] Coello Coello, C. A., Christiansen, A. D. & Aguirre, A. H. (1997), Automated design of combinational logic circuits using genetic algorithms, *en* D.G. Smith, N.C. Steele & R.F. Albrecht, eds, 'Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms ICANNGA '97', University of East Anglia, Springer-Verlag, Norwich, England, pp. 335-338.

[24] Grefenstette, J. J. (1993), Deception Considered Harmful, *en* L.D. Whitley, ed., 'Foundations of Genetic Algorithms 2', Morgan Kaufmann, San Mateo, California, pp. 75-91.

[25] Goldberg, D. E., Korb, B. & Deb, K. (1989), 'Messy genetic algorithms: Motivation, analysis, and first results', *Complex Systems* **3**, pp. 493-530.

[26] Goldberg, D. E., Deb, K. & Korb, B. (1990), 'Messy genetic algorithms revisited: Studies in mixed size and scale', *Complex Systems* **4**,415-444.

[27] Goldberg, D. E., Deb, K. & Korb, B. (1991), Don't worry, be messy, *en* R.K. Belew & L.B. Booker, editors, 'Proceedings of the Fourth International Conference on Genetic Algorithms', University of California, San Diego, Morgan Kaufmann Publishers, San Mateo, California, pp. 24-30.

[28] Mitchell, M. (1996), *An Introduction to Genetic Algorithms*, The MIT Press, Cambridge, Massachusetts.

[29] Chowdhury, M. M. & Li, Y. (1996), Messy Genetic Algorithm Based New Learning Method for Fuzzy Controllers, *en* 'Proceedings of the IEEE International Conference on Industrial Technology', IEEE Service Center, Shangai, China, http://eeapp.elec.gla.ac.uk/~chy/publications.html .

[30] Kajitani, I., Hoshino, T., Iwata, M. & Higuchi, T. (1996), Variable length chromosome GA for Evolvable Hardware, *en* 'Proceedings of the 1996 IEEE International Conference on Evolutionary Computation', Nagoya University, IEEE Service Center, Nagoya, Japan, pp. 443-447.

[31] Halhal, D., Walters, G.A., Ouazar, D. & Savic, D.A. (1997) , 'Water Network Rehabilitation with a Structured Messy Genetic Algorithm', *Journal of Water Resources Planning and Management*, ASCE **123**(3).