Resolución de Problemas Criptoaritméticos Utilizando Algoritmos Genéticos

Mónica Mounier[1], Facundo Aguirre[1], y Matías Barboza[1]

[1] Alumnos de 5[to] año de Ingeniería en Informática, Cátedra Inteligencia Artificial 2, Universidad Gastón Dachary, Av. López y Planes esq. Av. Jauretche, Posadas, Misiones, Argentina
{monicamounier, ignaciofacundoa, matias.mbz}@gmail.com

Resumen. Los problemas criptoaritméticos son problemas de satisfacción de restricciones (CPS) que generan un gran espacio de estados. Son rompecabezas en los que las letras del alfabeto deben ser remplazadas por dígitos, teniendo en cuenta para ello una serie de restricciones. Resolver este tipo de problemas mediante un proceso de razonamiento lógico, es una tarea difícil para el común de la gente, más aún cuanto mayor es su complejidad. Es por ello que implementar un mecanismo de resolución asistida por computadora es una opción muy adecuada para este tipo de problemas. En éste trabajo se propone la implementación de un Algoritmo Genético para resolver el problema planteado, junto con operadores de Selección Elitista, Cruza y Mutación que se adapten a las características específicas del mismo. Los resultados obtenidos muestran que esta estrategia es adecuada en la resolución de problemas criptoaritméticos complejos.

Palabras Claves: Algoritmos Genéticos, Operadores Genéticos, Problemas Criptoaritméticos.

1 Introducción

Los problemas criptoaritméticos son problemas de satisfacción de restricciones (CPS). Son rompecabezas en los que las letras del alfabeto deben ser remplazadas por dígitos, teniendo en cuenta para ello una serie de restricciones [1]. Resolver este tipo de problemas mediante un proceso de razonamiento lógico es una tarea difícil para el común de la gente, más aún cuanto mayor es su complejidad en cuanto a la cantidad de variables del problema, número de operandos y diversidad de los operadores.

Otra alternativa es resolver el problema de forma determinista, donde la cantidad de combinaciones posibles para 10 dígitos, por tratarse de permutaciones sin repetición, será de (1):

$$cp = \frac{10!}{(10 - ld)!} \quad (1)$$

Donde:
cp = Cantidad de combinaciones posibles del problema criptoaritmético planteado.
ld = Cantidad de letras diferentes del problema criptoaritmético planteado.

En el peor escenario posible, con ld igual a 10, se obtienen 3.620.800 combinaciones posibles, entre las cuales puede haber una ó varias soluciones al problema, y en el peor de los casos ninguna solución.

Actualmente, existen numerosos trabajos que plantean resolver este tipo de problemas mediante la implementación de un AG como ser Algoritmos Genéticos Paralelos [2], Algoritmos Genéticos con Manejo de Restricciones [3], Algoritmos Evolutivos para Problemas de Optimización con Parámetros Restringidos [4].

En el presente trabajo, se busca resolver problemas criptoaritméticos con diversos operadores (suma, resta, multiplicación y división), y un mínimo de tres operandos de al menos 5 dígitos. Para ello, se propone la resolución de problemas criptoaritméticos mediante la utilización de un AG, por medio de la implementación de operadores de Selección Elitista combinando con Selección por Ranking [5], Cruza Cíclica [6] y una variante propuesta del Operador de Mutación Simple [6] denominada Operador de Mutación Cíclica Adaptativa. En la sección 2 se presenta una descripción de los problemas Criptoaritméticos y Algoritmos Genéticos. En la sección 3 se describe el AG propuesto. En la sección 4 se muestran los resultados obtenidos de las pruebas realizadas. Finalmente, en la sección 5 se presentan las conclusiones del trabajo.

2 Marco Teórico

2.1 Problemas Criptoaritméticos

Un problema criptoaritmético es un rompecabezas en el que las letras del alfabeto deben ser remplazadas por dígitos. El objetivo es determinar los dígitos que se asignarán a las letras contenidas en el problema, cumpliendo las restricciones previstas por la aritmética y la condición adicional de que dos letras no pueden tener el mismo valor numérico (comprendido entre 0 y 9) [1]. En (2) se presenta uno de los problemas criptoartméticos más conocidos [6].

$$\begin{array}{r} \text{SEND} \\ +\,\text{MORE} \\ \hline \text{MONEY} \end{array} \qquad (2)$$

Una vez asignados los dígitos en (2), se obtiene el resultado de la operación aritmética, el cual es comparado con el valor numérico resultante de las asignaciones de dígitos realizadas aleatoriamente. Si ambos resultados coinciden, se ha encontrado una solución al problema, caso contrario, se vuelven a asignar dígitos a las letras.

2.2 Algoritmos Genéticos

Un AG simula la evolución de una población de individuos, mediante un proceso iterativo aplicado sobre un conjunto de estructuras, donde cada una de dichas estructuras está compuesta de características que definen la aptitud del individuo en su entorno. La población evoluciona de generación en generación mediante la recombinación de sus integrantes, la mutación de algunas características elegidas al azar, y la selección de las estructuras más aptas [5].

En la resolución de problemas, un individuo representa una posible solución a un problema dado, y la aptitud del mismo es una medida de cuan buena es la solución para el problema. La generación de la población inicial se realiza generalmente de forma aleatoria. Luego, cada generación se crea a partir de la generación anterior tras la aplicación de tres operadores básicos: selección, cruza y mutación. El proceso iterativo sigue hasta que el AG cumple con la condición de parada, la cual podría ser que el individuo presenta una aptitud suficientemente buena, se alcanza una cantidad máxima de generaciones, ó se cumple una condición específica del problema [5][8].

Un AG simple puede describirse como sigue [7]:

Algoritmo 1. Algoritmo Genético Simple

```
01. procedimiento EA {
02.    t = 0;
03.    inicializar_poblacion P(t);
04.    evaluar P(t);
05.    hasta (hecho) {
06.      t = t + 1;
07.      seleccionar_padres P(t);
08.      recombinar P(t);
09.      mutar P(t);
10.      evaluar P(t);
11.      sobrevivir P(t);
12.    }
13.  }
```

3 Descripción de la Solución Implementada

3.1 Definición del Cromosoma

La primera parte de la solución de un problema utilizando AG es definir la forma de representar los individuos. Puesto que en este problema se presentan números enteros de 0 a 9, se utiliza un vector de longitud 10, con índices de 0 a 9, para representar los individuos de la población. Para asignar un dígito a una letra, se ubica a la misma aleatoriamente en una posición del vector, y el índice de esta posición será el dígito asignado. Únicamente se utiliza una copia de cada letra y las posiciones sin letras quedan vacías [2]. En la Fig. 1 se puede ver un ejemplo de asignaciones para las letras del problema presentado en (2), donde se ha asignado el número 2 a la letra M.

S	O	M		E	R	Y	N		D
0	1	2	3	4	5	6	7	8	9

Fig. 1. Un ejemplo de individuo

Esta representación fue utilizada debido a las ventajas en la implementación del AG, ya que a partir de su estructura se conoce la asignación de los dígitos a las letras del problema, así también como los dígitos disponibles para posibles asignaciones.

3.2 Función de Aptitud

Para determinar la aptitud del individuo, se tiene en cuenta el resultado del cálculo aritmético según los dígitos asignados a las letras de los operandos, en contraste con el valor numérico obtenido mediante el remplazo de las letras contenidas en el resultado del problema dado, por los dígitos asignados de aleatoriamente.

La función de aptitud propuesta consiste en comparar los dígitos de los resultados mencionados, incrementando la aptitud cuanto más diferencias existan, iniciando en el dígito de la unidad (posición 0), y finalizando al comparar todos los dígitos de ambos resultados. Además, es penalizado el incumplimiento de las restricciones aritméticas aumentando el valor de la aptitud del individuo (3).

$$aptitud = \sum_{i=0}^{m-1} |x_i - y_i| \times (m - i) + \sum_{j=m}^{n-1} y_j \times 10^{(j-m)+2} + r \times c \qquad (3)$$

Donde:
$aptitud$ = Aptitud del individuo. Cuando esta se acerca a cero, mejor es el individuo.
m = Número de dígitos del resultado de menor longitud.
n = Número de dígitos del resultado de mayor longitud.
x_i = Es el i-ésimo digito del resultado de menor longitud.
y_i = Es el i-ésimo digito del resultado de mayor longitud.
r = Penalización por incumplimiento de una restricción aritmética. Su valor es 5790.
c = Cantidad de restricciones aritméticas no cumplimentadas por el individuo.

La función aptitud descrita en la ecuación (3) logra que la diferencia encontrada en la unidad de los resultados comparados tenga mayor peso que en la decena, centena, etc., logrando así empeorar la aptitud del individuo que posea menor probabilidad de ser una solución al problema, dadas las diferencias existentes en el resultado esperado. Además, el incumplimiento de restricciones aritméticas desmejora notablemente la aptitud del individuo.

Para el ejemplo de la Fig. 1, el resultado de la operación aritmética mediante el remplazo de las letras por los dígitos del problema planteado (2) es 0479(SEND) + 2154(MORE) = 2633, y el resultado al remplazar MONEY por los dígitos asignados aleatoriamente es 21746(MONEY). Entonces, la aptitud del individuo es igual a 6008.

3.3 Operador de Selección

Para la selección de individuos, se propone la utilización del operador de Selección Elitista en conjunto con el operador de Selección por Ranking Lineal, con R_{min} (número mínimo de copias) igual a 0, para no copiar los peores individuos [5]. En el operador de Selección por Ranking se introdujo una modificación en el método de ordenamiento de individuos, porque fue creado para problemas de maximización, y la función de aptitud planteada (3) tiene por objetivo la minimización (orden ascendente).

Durante el proceso de selección elitista se consideran los individuos a los cuales se han asignado una o más copias en el Ranking, y se los elije de forma tal que se alcance la mayor diversidad posible hasta obtener la cantidad de individuos deseada, correspondiente al 10% de la población, ó al porcentaje indicado por el usuario.

3.4 Operador de Cruza

Debido a la naturaleza del problema criptoaritmético, se debe evitar que diferentes genes posean la misma letra, por lo cual se utiliza el operador de Cruza Cíclica [6]. Para la selección de los individuos a cruzar, se utiliza el método proporcional de Selección por Ruleta [5], al cual se introdujo una modificación en la determinación de la cantidad de ranuras correspondientes a cada individuo, ya que este método es utilizado para problemas de maximización, opuesto al objetivo de minimización de la función de aptitud propuesta en el presente trabajo. Para el cálculo de la cantidad de ranuras asignadas a un individuo, se tienen en cuenta la razón inversa de la aptitud y la aptitud del peor individuo (4). La cantidad total de ranuras es igual a 10^n.

$$cr = \frac{1}{aptitud} \times 10^n \tag{4}$$

Donde:
cr = Cantidad de ranuras asignadas a un individuo.
n = Cantidad de dígitos de la peor aptitud de la población (mayor valor numérico).

Durante el proceso de cruza, se generan dos hijos a partir de los individuos seleccionados, donde si los hijos son iguales a los padres, se repite hasta 10 veces el proceso, después de ello se elijen padres diferentes para cruzar. Este proceso continúa hasta obtener el porcentaje deseado de individuos de cruza de la población, inicialmente representa el 90% de la misma, ó el porcentaje indicado por el usuario.

La selección de individuos a cruzar mediante ruleta posibilita que los individuos con mejor aptitud tengan mayor probabilidad de ser elegidos. En las pruebas realizadas se obtuvieron buenos resultados con el 90% de individuos mediante cruza.

3.5 Operador de Mutación

En el operador de mutación se implementa una variante de la Mutación Simple [6], denominada Mutación Cíclica Adaptativa. La selección de individuos a mutar se realiza del mismo modo que en la cruza (ver subsección 3.4). La variante planteada, realiza una cierta cantidad de cambios en el individuo teniendo en cuenta para ello la su aptitud, y cuanto peor es la misma, mayor es la cantidad de cambios a realizar (5).

$$cc = l + redondeo\left(\frac{aptitud}{10^l}\right) \tag{5}$$

Donde:
cc = Cantidad de cambios a realizar en el individuo.
l = Cantidad de dígitos de la aptitud.

Para aplicar los cambios en el individuo, se obtiene un número aleatorio (entre 0 a 9) a fin de determinar el gen a mutar, por ejemplo, si la cantidad de cambios a realizar es igual a 2, entonces se genera una secuencia de 3 números aleatorios. La secuencia de aleatorios debe ser diferente en sus extremos, y a su vez cada aleatorio consecutivo debe ser diferente. A partir de esta secuencia se realiza la mutación de los genes, donde el gen de la posición x_i se coloca en la posición x_{i+1}, y así sucesivamente, hasta

llegar al último número de la secuencia x_n. El último gen ubicado en la posición x_n se coloca en la posición x_i, cerrando de esta forma el ciclo de mutación.

La adaptación del operador influye en la cantidad de individuos a mutar, la cual varía según el comportamiento del mejor individuo de la generación, partiendo inicialmente en un 0% de individuos, y luego, cada 10 generaciones se verifica si se produce una mejora el mismo, aumentando un 1% si no se producen cambios. Esto se realiza hasta llegar al porcentaje máximo de mutación del 10%. En la medida que aumenta el porcentaje de individuos a mutar, disminuye el porcentaje de individuos a cruzar.

Mediante la selección por ruleta los individuos con mejor aptitud tienen mayor probabilidad de ser elegidos para la mutación, así también, el operador de mutación propuesto aumenta la diversidad de la población en la medida en que sea necesaria.

3.6 Criterios del AG

A continuación se definen varios criterios del AG propuesto:

- **Criterio de Inicialización:** La población inicial se crea aleatoriamente, a partir de la cantidad de individuos indicada por el usuario, siendo el valor por defecto de 12500 individuos, debido a los buenos resultados obtenidos con esta cifra.
- **Tratamiento de Individuos no Factibles:** Son individuos no factibles aquellos en los que se asigna un denominador 0 en una división, el minuendo es menor al sustraendo, y aquellos en los que el denominador es mayor al numerador (excepto cuando este es igual a 0). A estos individuos se asigna la peor aptitud disponible en la población, reduciendo la probabilidad de que pasen a la siguiente generación.
- **Criterio de Parada:** Se plantean tres criterios de parada, el primero consiste en encontrar un individuo que sea solución al problema, es decir, con aptitud igual a 0. El segundo criterio evalúa el histórico del mejor individuo en cada generación, y si durante una determinada cantidad de generaciones no mejora su aptitud, se detiene el AG. El tercer y último criterio, consiste en llegar a la cantidad de generaciones especificadas por el usuario, por defecto no hay límite de generaciones.

3.7 Algoritmo Genético Propuesto

El AG propuesto, denominado CryptoAG, se inicia cuando el usuario indica el problema criptoaritmético a resolver. Luego, se valida la sintaxis del problema ingresado, donde si es correcta, se genera la población inicial teniendo como parámetro la cantidad de individuos definidos por el usuario, con 12500 individuos por defecto. Una vez generada la población inicial, se calcula y evalúa la aptitud de todos los individuos, donde si su aptitud es igual a 0, el AG se detiene informando la solución encontrada. Posterior a ello, se modifican las aptitudes de los individuos no factibles (ver subsección 3.6). A partir de la población obtenida, se procede con la Selección Elitista, Cruza Cíclica y Mutación Cíclica Adaptativa. El AG continúa hasta que se cumpla alguna de las condiciones de parada (ver subsección 3.6), como se muestra en la Fig. 2.

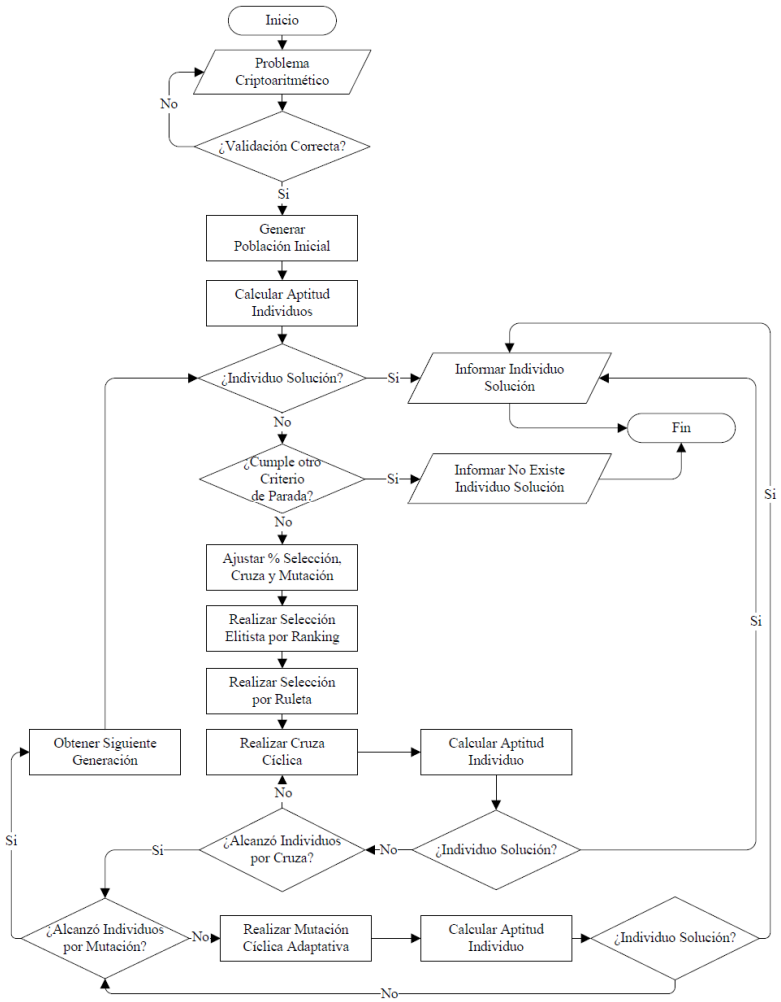

Fig. 2. Diagrama de Flujo de CryptoAG

En el Diagrama de Flujo de CryptoAG (Fig. 2), se puede observar el funcionamiento básico del AG, donde cabe destacar el procedimiento de ajuste de los porcentajes de selección, cuza y mutación, el cual posibilita adaptar la aplicación de los operadores, según el análisis de la evolución de la población, aumentando o disminuyendo dicho porcentaje según sea necesario. El funcionamiento de los demás componentes del diagrama ha sido ampliamente explicado en las subsecciones anteriores.

3.8 Tecnologías Utilizadas para la Implementación del AG

La implementación de CryptoAG fue realizada en el lenguaje de programación C, bajo el S.O. Debian GNU/Linux 7.0, y se utilizaron los siguientes componentes:

- **libc:** Librería estándar de C de GNU.
- **GTK+ v3:** Librería utilizada para la implementación de la interfaz gráfica.
- **gnuplot:** Programa utilizado graficar el comportamiento del AG.
- **gnuplot_i.c:** Funciones auxiliares para facilitar la interacción con gnuplot.
- **Flex:** Programa utilizado para la implementación del analizador léxico para la definición los tokens posibles en un problema criptoaritmético.
- **GNU Bison:** Programa utilizado para la implementación del analizador sintáctico necesario para validar los problemas criptoaritméticos ingresados por el usuario.
- **mtwist.c:** Implementa el algoritmo MercenneTwister (números pseudoaleatorios).

4 Simulación y Resultados Obtenidos

CryptoAG fue testeado con diferentes tipos problemas criptoaritméticos para evaluar su comportamiento. Los resultados se obtuvieron en una computadora con un procesador Intel Core i5 1,7 GHz, 4 GB de memoria. Para analizar su comportamiento, se plantearon distintos problemas con distinta cantidad y tipo de operaciones, cantidad de variables, cantidad de operandos y tamaño de la población. Cada uno de los problemas fue testeado 100 veces y los resultados obtenidos se muestran en la Tabla 1.

Tabla 1. Resultados obenidos con el CryptoAG para 100 corridas

Problema	V	Ind	G_{min} (ms)	G_{max} (ms)	G_{med} (ms)	T_{min} (ms)	T_{max} (ms)	T_{med} (ms)
ONE+ONE = TWO	5	500	0	74,2	3,33	0	129	5,10
(271 + 271 = 542)		1000	0	21	0,51	0	79,1	4,45
ABC-CA = ADE	5	500	0	2	0,84	0	2,1	0,11
(854 - 48 = 806)		1000	0	1	0,02	0	1,4	0,18
ABC+CDEF = CCAC	6	1800	0	2	0,25	1	16,8	2,63
(125 + 5390 = 5515)		3600	0	0,7	0,06	1,4	24,5	2,82
ABC+(CDE*BFE) = GBCBC	7	2940	0	14,7	3,10	1,5	265	56,7
(923 + (340 * 210) = 72323)		5880	0	493	6,65	2,1	2307	140
SEND+MORE = MONEY	8	3840	0	4,9	2,2	1,4	137	56,7
(5849 + 0638 = 06487)		7680	0	3,5	1,54	2,8	312	128
CAT + DOG = PETS	9	6250	0	1,4	0,46	2,1	76,3	15,8
(947 + 623 = 1570)		12500	0	1,3	0,16	4,9	163	19,8
PLAYS +WELL = BETTER	10	6250	0	8,4	3,54	7,7	482	194
(97426 + 8077 = 105503)		12500	0	5,6	2,78	7	1192	494

Donde: **V**: Número de variables, **Ind**: Número de individuos, $\mathbf{G_{min}}$: Número mínimo de generaciones al encontrar una solución, $\mathbf{G_{max}}$: Número máximo de generaciones al encontrar una solución, $\mathbf{G_{med}}$: Número medio de generaciones al encontrar una solución, $\mathbf{T_{min}}$: Tiempo mínimo en ms de ejecución del AG para encontrar una solución, $\mathbf{T_{max}}$: Tiempo máximo en ms de ejecución del AG para encontrar una solución, $\mathbf{T_{med}}$: Tiempo medio en ms de ejecución del AG para encontrar una solución.

En la Tabla 1 se puede observar que al aumentar la cantidad de individuos en la población, disminuye el promedio de generaciones en encontrar una solución, no obstante, el tiempo promedio aumenta en forma proporcional al tamaño de la misma. Asimismo, se puede observar en esta tabla, que el número medio de generaciones en encontrar una solución en las distintas corridas para los problemas planteados no supera las 6,65 generaciones, y un tiempo medio máximo de 494 milisegundos.

Durante las pruebas realizadas, se implementaron distintas variantes de los operadores propuestos en el trabajo, a partir de lo cual se pudo observar que la configuración recomendada es la que generaba mejores resultados frente a las demás, por converger más rápidamente a la resolución de los problemas testeados (Tabla 1).

Con el propósito de comparar el rendimiento de CryptoAG, se ha adaptado una rutina, denominada Backtraking [9][10], que resuelve problemas criptoaritméticos de forma determinista, la cual consiste en generar todas las combinaciones posibles hasta encontrar una solución al problema dado. Los problemas indicados en la Tabla 1 fueron testeados una sola vez y los resultados obtenidos se presentan en la Tabla 2.

Tabla 2. Resultados obenidos con CryptoAG y Backtraking

Variables	$\mathbf{T_{med}}$**CryptoAG** (ms)	**T Backtraking** (ms)
5	5,10	3220
6	2,63	6960
7	56,7	5490
8	56,7	5890
9	15,8	2240
10	194	9850

Donde: $\mathbf{T_{med}}$**CryptoAG**: Tiempo medio en ms de ejecución del AG en encontrar una solución, **T Backtraking**: Tiempo en ms de ejecución del programa Backtraking en encontrar una solución.

En la Tabla 2 se puede ver que los resultados obtenidos mediante CryptoAG han mostrado ser muy superiores a los obtenidos a través del programa Backtraking, debido a que el tiempo medio de respuesta de CryptoAG para problemas de 6 variables fue, en el mejor de los casos, 2646 veces menor, y 50 veces menor en los casos más desfavorables, siendo estos los problemas con 10 variables.

5 Conclusiones

Los resultados obtenidos demuestran que es adecuado utilizar algoritmos genéticos para resolver problemas criptoaritméticos, debido a exploran el espacio de estados convergiendo rápidamente a una de las soluciones del problema, siempre que exista.

La función de aptitud propuesta en este trabajo es adecuada para este tipo de problemas, dado que permite evaluar correctamente a los individuos de la población según la proximidad a una solución válida.

CryptoAG ha mostrado mejor performance que el programa Backtraking, no obstante, se ha observado un comportamiento inestable en algunos problemas que tienen una única solución y muchas variables.

Los operadores implementados han posibilitado el buen rendimiento del AG, ya que mediante la técnica de Selección Elitista combinada con la Selección por Ranking ha permitido conservar a los individuos más aptos para pasar a la siguiente generación. Mediante el operador de Cruza Cíclica se observó que en muchos casos, ha dado como resultado de la cruza un individuo solución al problema planteado. Finalmente, el operador de Mutación Cíclica Adaptativa ha logrado aumentar oportunamente la diversidad de la población.

Una alternativa para mejorar la performance en la resolución de todos problemas criptoaritméticos, en especial de aquellos con una única solución y muchas variables, es tener en cuenta las restricciones específicas del problema planteado por el usuario para todas las operaciones aritméticas presentes y sus posibles combinaciones a la hora de evaluar la factibilidad del individuo. De esta forma, en todo momento se generarían únicamente individuos que sean posibles soluciones al problema, lo que si bien posibilitaría converger más rápido a una solución, en caso de que la hubiere, también aumentaría significativamente la complejidad del AG.

6 Bibliografía

1. Soni, H., Arora, N.: Solving Crypt-Arithmetic Problems Via Genetic Algorithm. Jmacademy of IT &Management, vol. 1, p. 13 (2011)
2. Abbasian, R., Mazloom, M.: Solving Cryptarithmetic Problems Using Parallel Genetic Algorithm. Second International Conference on Computer and Electrical Engineering (2009)
3. Deb, K.: An Efficient Constraint Handling Method for Genetic Algorithms. Computer Methods in Applied Mechanics and Engineering 184 (2000)
4. Michalewicz, Z., Schoenauery, M.: Evolutionary Algorithms for Constrained Parameter Optimization Problems (1996)
5. García Martínez, R., Servente, M., Pasquini, D.: Sistemas Inteligentes. Nueva Librería SRL. 149-199 (2003)
6. Goldberg, D.E.: Genetic Algorithms in Search, Optimation& Machine Learning, Addison-Wesley, Massachusetts (1989)
7. Spears, W.M., De Jong, K.A., Back, T., Fogel, D.B., De Garis, H.: An Overview of Evolutionary Computation. European Conference on Machine Learning, Vienna, Austria (1993)
8. Blickle, T., Thiele, L.: A comparison of selection schemes used in genetic algorithms. Technical Report 11, Computer Engineering and Communication Networks Lab. Swiss Federal Institute of Technology (1995)
9. Russell, S.; Norvig, P.: Artificial Intelligence, A Modern Approach. Prentice Hall, Third Edition. 337-345 (2010)
10. Write a C program to print all permutations of a given string, http://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/