Utilización de algoritmos de enjambre para resolver problemas criptoaritméticos

María Laura Acuña\*, Marcelo Espinoza\*, Cecilia María Luciana Gómez\*

\*Alumnos de 5to año de la carrera ingeniería en sistemas de información, Facultad Regional Resistencia, Universidad Tecnológica Nacional – French 414 – Resistencia, Chaco, Argentina. CP 3500.

{Marilau\_ml, marceloespinoza00, Cecilia.mlgz}@gmail.com

**Abstract.** Los algoritmos de enjambre son utilizados no solamente para resolver problemas que se presentan en la naturaleza, sino que tienen múltiples aplicaciones. En este trabajo, nos centramos en los problemas criptoaritméticos de sumas o restas, utilizando para ello un tipo de algoritmo de enjambre, específicamente el algoritmo firefly, tal algoritmo fue adaptado para que se ajuste al problema planteado. La resta es una adaptación de la suma.

**Keywords:** Algoritmos genéticos, criptoaritmética, algoritmos de ejambre, firefly

1. Introducción

Los problemas criptoaritméticos son puzzles donde las letras son reemplazadas por números, son problemas de restricciones (Mounier – Aguirre – Barboza) [2], que si se intentan resolver manualmente generan un gran espacio de búsqueda, dificultando así la generación de la solución. Existen varias alternativas para encontrar la solución, en (Mounier – Aguirre – Barboza) [2] se plantea una solución utilizando algoritmos genéticos con operaciones de cruza y mutación, en Ishaque y otros [3] se plantea una solución utilizando también algoritmos genéticos, con operaciones de mutación, y se demuestra la mejora en performance respecto a la búsqueda heurística. En SARAEI [6] se describe cómo solucionar el problema del viajante aplicando el algoritmo propuesto por Yang [1]. Por tales motivos se plantea una solución implementando, con algunas adaptaciones al algoritmo planteado por Yang [1], redefiniendo una función distancia y la función acercamiento.

En la sección 2 se presenta una introducción a los algoritmos de enjambre, especificando el algoritmo firefly. En la sección 3 se describe el modelo planteado. En la sección 4 se muestran los casos de prueba y los resultados obtenidos. Finalmente, en la sección 5 las conclusiones del trabajo.

* 1. Problema Planteado

El problema consiste en resolver puzzles criptoaritméticos, de sumas o restas, con la implementación del algoritmo firefly [1].

Este problema presenta las restricciones siguientes:

* + La cantidad de letras no repetidas de los operandos a sumar (o restar) no debe ser mayor a 10.
  + Cada letra se identifica con un único número y ese número representa a una única letra.
  + El resultado deber ser acorde a la suma algebraica de los operandos.

El número mínimo de caracteres a ingresar por operando debe ser mayor o igual 5.

El problema planteado presenta la complejidad de que, cómo utiliza una función discreta, hay que resolverlo mediante la utilización de un algoritmo funciones continuas para encontrar máximos locales.

1. ¿Qué son los algoritmos de enjambre?

Los algoritmos de enjambre simulan el comportamiento que tienen ciertas especies que se organizan grupalmente para subsistir. Algunos de los algoritmos de enjambres más conocidos son “La optimización de colonias de hormigas” – Dorigo [8], “La optimización de colonias de abejas” – De los Cobos [9], y entre estos tipos de algoritmos se encuentra el desarrollado por Yang [1].

La inteligencia de enjambre estudia el comportamiento colectivo compuesto por muchos individuos interactuando localmente y con su entorno – Molina [5].

En un sistema de optimización por enjambre de partículas, la búsqueda se realiza utilizando una población de partículas que corresponden a los individuos, cada uno de los cuales representa una solución candidata al problema. Las partículas cambian su estado al moverse a través del espacio de búsqueda hasta que se ha encontrado un estado relativamente estable, la solución.

* 1. Algoritmo de luciérnagas

El algoritmo de luciérnagas (firefly algorithm en inglés) conocido como FA, está basado en el comportamiento de las luciérnagas en la naturaleza.

El algoritmo se basa en tres reglas básicas:

* + Todas las luciérnagas son unisexuales y se sienten atraídas por otras luciérnagas, independientemente de su sexo.
  + El grado de atracción de una luciérnaga es proporcional a su brillo, por lo tanto para cualquier par de luciérnagas, la que menos brillo tiene se moverá hacia la que más brillo tiene.
  + En el caso de que las dos luciérnagas tengan el mismo brillo, se moverán aleatoriamente.

El brillo de una luciérnaga se determina por el valor de la función objetivo.

El algoritmo de luciérnagas actualmente tiene diferentes variantes, los cuales algunos de ellos son: Adaptive Firefly Algorithm (AdaFa), Dis-crete Firefly Algorithm (DFA) y Chaotic FA.

El campo de aplicación del algoritmo de luciérnagas es bastante amplio, pudiendo destacarse Compresión de imagen digital y procesamiento de imagen, diseño de antenas, etc. (Yang) [10].

1. Funcionamiento del Algoritmo desarrollado

Los valores de entrada se colocan en un vector que sirve como base para el desarrollo de todo el algoritmo. De esta manera, si se ingresa

APPLE+LEMON=BANANA

El vector de inicio se forma tomando primero las unidades de los operadores y el resultado, luego las decenas y así sucesivamente, luego de acomodarlo y quitarle los elementos repetidos (se completa con “-“ hasta las diez elementos dentro del vector) queda:

LETRAS DE ENTRADA: e n a l o p m b - -

POSICION: 0 1 2 3 4 5 6 7 8 9

Las posiciones de las letras sirven para controlar las posibles soluciones.

Luego, se crean una matriz de vectores de diez números aleatorios cada uno, la población inicial. Cada vector representa una luciérnaga, la cual se va adaptando para llegar a ser una solución.

Por cada luciérnaga, se extraen los valores, APPLE, LEMON y BANANA, que corresponden a su posición en el vector de inicio, y se calcula la suma algebraica entre APPLE y LEMON, el brillo de la misma, que es obtenido entre la suma algebraica y la correspondencia del vector resultado en el vector inicio, cuanto más alto sea el brillo, más cerca se encuentra de ser una solución.

En la figura 1, se describe el desarrollo general de la aplicación, y en la figura 2, se describe el funcionamiento particular de la función aplicar algoritmo.

* 1. Función Distancia

La función distancia definida (algoritmo 1), recibe como parámetros dos vectores que corresponden a la suma algebraica de dos luciérnagas diferentes y devuelve un valor entre uno y diez.

Algoritmo 1: Función Distancia.

1. function distancia($X1, $X2) {
2. $i = count($X1);
3. $d = 0;
4. for ($j=0; $j < $i; $j++) {

05. if ($X1[$j] = $X2[$j]) {

06. $d = $d + 10\*$j;

07. }

08.}

09.$d = fmod($d, 10);

10.return $d;

11.}

* 1. La función objetivo

Se definió la función objetivo como “obtenerBrillo” (ver algoritmo 2) la que recibe como parámetro la suma algebraica de los valores que corresponden a los operadores de entrada, y el resultado que corresponden a los valores del operando ingresado como resultado.

Su funcionamiento consiste en devolver un valor de brillo que depende de la cantidad de elementos que tenga el vector de resultado ingresado, y las coincidencias que tenga con el vector suma, comparando posición a posición.

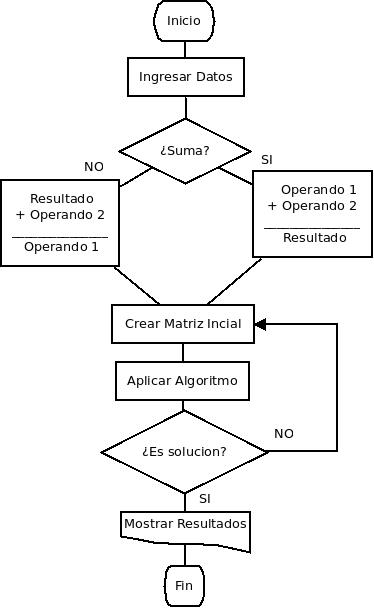
Algoritmo 2: Función objetivo*.*

1. Public function obtenerBrillo($suma,$resultado){
2. $res=$this->intToArray($resultado);
3. $sum=0;
4. $i=(sizeof($res)-1);
5. $j=(sizeof($suma)-1);
6. $counter=0;
7. if(sizeof($res)<sizeof($suma)){
8. $counter=sizeof($res);
9. }else{
10. $counter=sizeof($suma);
11. }
12. for ($k = ($counter-1); $k >=0; $k--) {
13. if($res[($i)]==$suma[($j)]){
14. $i--;
15. $j--;
16. $sum++;
17. }
18. }
19. return $sum;
20. }
    1. Function Movimiento.

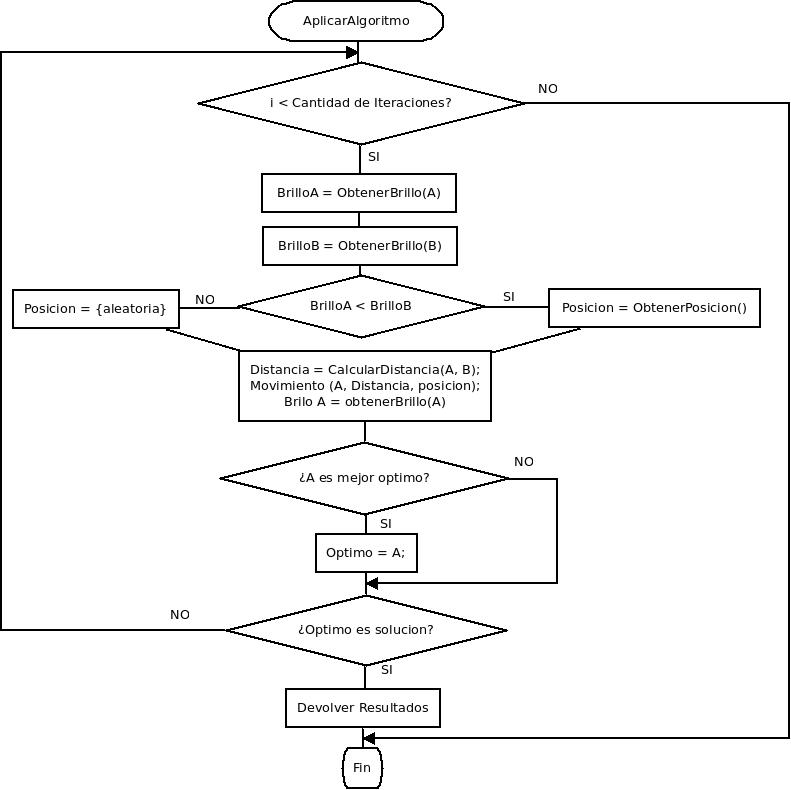
La función movimiento devuelve el nuevo valor que tendrá la posición seleccionada y está compuesta por 3 términos. El primero es el valor actual, el segundo es el que está afectado por la distancia entre luciérnagas, el cual está formulado para que en aquellos casos en que el brillo de las luciérnagas sean iguales, la distancia sea cero y dicho término se anule, afectando al nuevo valor solamente el tercer término que devuelve un valor aleatorio.

#### Algoritmo 3: Función movimiento*.*

1. function movimiento(luciernaga,distancia,pos){
2. beta = e^(-1)\*(distancia);
3. epsilon = 1;
4. alfa = rand(0,1);
5. actual = luciernaga[pos];
6. nuevo=luciernaga[pos]+(1-beta)+alfa\*epsilon;
7. nuevo = extraer\_unidad(nuevo);
8. if (nuevo>=0 and nuevo<10){
9. luciernaga=sinRepetidos(luciernaga);
10. luciernaga[pos]=nuevo;
11. }else{
12. $X1[$pos]=$elementoActual;
13. }
14. return $X1;
15. }



**Fig. .** Diagrama de Flujo de la estructura de la solución planteada



**Fig. .** Diagrama de Flujo del funcionamiento del algoritmo para encontrar una solución óptima

1. Resultados obtenidos

Para la evaluación de la solución propuesta se consideraron diferentes problemas criptoaritméticos que cumplen con las restricciones planteadas en la sección 1. A su vez, se seleccionaron para las pruebas problemas con diferentes características, como se ve en la Tabla 1.

**Tabla 1.** Características de los problemas criptoaritméticos empleados en las pruebas.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Entrada** | **Solución** | **Cant.**  **Soluciones Factibles** | **Cant. de Dígitos Distintos** | **Mayor Cant.**  **Dígitos Operando** |
| APPLE+LEMON=BANANA | 67794+94832=162626 | Una | 8 | 6 |
| MENTAL+HEALTH=MATTERS | 123408+920849=1044257 | Una | 8 | 7 |
| APPLE+FLOPPY=REBOOT | 74431+538449=612880 | Una | 10 | 6 |
| PROFE+PROFE=SABER | 18274+18274=36548 | Muchas | 8 | 5 |

Además, se varía la operación entre suma y resta, la cantidad de luciérnagas y la cantidad de iteraciones iniciales del algoritmo. En la tabla 2 se muestran los resultados obtenidos para 10 corridas por cada caso de prueba. Por otra parte, se utiliza una configuración inicial de 10 iteraciones. Es decir, el algoritmo intenta converger a la solución hasta con 10 iteraciones, pasadas las 10 iteraciones, explora otro espacio de posibles soluciones.

**Tabla 2.** Resultados obtenidos de los casos de prueba

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Entrada** | **Cant.**  **Luciérnagas** | | **Cant.**  **Corridas con Solución** | | **Nro.**  **Iteraciones Promedio** | **Tiempo Rta. Promedio (seg)** | |
| APPLE+LEMON=BANANA | 50 | 10 | | 599,8 | | 255,3 | |
| APPLE+LEMON=BANANA | 100 | 10 | | 138,5 | | 184,4 | |
| APPLE+LEMON=BANANA | 150 | 10 | | 41,0 | | 109,9 | |
| MENTAL+HEALTH=MATTERS | 50 | 7 | | 656,8 | | 347,1 | |
| MENTAL+HEALTH=MATTERS | 100 | 10 | | 196,7 | | 357,0 | |
| MENTAL+HEALTH=MATTERS | 150 | 10 | | 43,1 | | 118,3 | |
| APPLE+FLOPPY=REBOOT | 50 | 10 | | 1112 | | 295,6 | |
| APPLE+FLOPPY=REBOOT | 100 | 10 | | 302,6 | | 358,9 | |
| APPLE+FLOPPY=REBOOT | 150 | 10 | | 254 | | 576,8 | |
| PROFE+PROFE=SABER | 50 | 10 | | 34,8 | | 21,5 | |
| PROFE+PROFE=SABER | 100 | 10 | | 10,2 | | 23,8 | |
| PROFE+PROFE=SABER | 150 | 10 | | 7,2 | | 9,2 | |
| BANANA-LEMON=APPLE | 50 | 10 | | 111,6 | | 32,8 | |
| BANANA-LEMON=APPLE | 100 | 10 | | 78,5 | | 159,9 | |
| BANANA-LEMON=APPLE | 150 | 10 | | 123,7 | | 493,4 | |
| MATTERS-HEALTH=MENTAL | 50 | 10 | | 878,1 | | 318,0 | |
| MATTERS-HEALTH=MENTAL | 100 | 10 | | 191,8 | | 230,0 | |
| MATTERS-HEALTH=MENTAL | 150 | 10 | | 115,3 | | 543,0 | |
| REBOOT-FLOPPY=APPLE | 50 | 10 | | 513,5 | | 157,0 | |
| REBOOT-FLOPPY=APPLE | 100 | 10 | | 219,2 | | 210,0 | |
| REBOOT-FLOPPY=APPLE | 150 | 10 | | 115,1 | | 281,0 | |
| SABER-PROFE=PROFE | 50 | 10 | | 31,4 | | 4,3 | |
| SABER-PROFE=PROFE | 100 | 10 | | 22,4 | | 7,1 | |
| SABER-PROFE=PROFE | 150 | 10 | | 17,2 | | 9,8 | |
| **PROMEDIOS GENERALES** | | | | | **242,2** | **212,6** |

Como pueden observarse en las pruebas presentadas en la Tabla 2, el algoritmo llega a una solución de forma más eficiente si el problema presenta más de una solución. Que, como se ve en la Tabla 1, es el caso de PROFE+PROFE=SABER.

Por otro lado, se realizaron pruebas variando el número de luciérnagas. Se observa que, en la mayoría de los casos, a mayor número de luciérnagas, decrementan las iteraciones, sin embargo, aumenta el tiempo de respuesta y viceversa. Esto se debe a que, al aumentar el número de luciérnagas, aumenta exponencialmente el número de comparaciones que el algoritmo debe efectuar hasta alcanzar la solución. La Tabla 3 presenta un resumen de las pruebas realizadas promendiando el número de iteraciones y el tiempo de respuesta obtenidos de acuerdo a la cantidad de luciérnagas.

**Tabla 3.** Resumen de pruebas por cantidad de luciérnagas

|  |  |  |
| --- | --- | --- |
| **Cantidad**  **Luciérnagas** | **Promedio de**  **Número de Iteraciones** | **Promedio de Tiempo**  **Respuesta.(seg)** |
| 50 | 492,25 | 178,9 |
| 100 | 144,99 | 192,8 |
| 150 | 89,58 | 266,3 |
| **Promedio general** | **242,27** | **212,6** |

Por otra parte, no se observa que la cantidad de dígitos o caracteres diferentes del caso de prueba afecte de alguna manera la performance del algoritmo. Lo mismo puede decirse de la cantidad de dígitos de los operandos, lo cual no hace que se dificulte o mejore el llegar a una solución.

En cuanto a resolver una operación u otra, en las pruebas se ven mejores resultados en tiempo de respuesta e iteraciones realizadas al realizar la resta. Sin embargo, estas diferencias no resultan significativas.

Finalmente, se realizaron pruebas variando el número de iteraciones, que es otro parámetro de configuración del algoritmo. La Tabla 4 muestra dichos resultados, en los que se puede observar que es preferible un número bajo de iteraciones del algoritmo para que sea eficaz.

**Tabla 4.** Pruebas realizadas variando la cantidad de iteraciones

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Cantidad de Luciérnagas** | 50 | 100 | 100 | 100 | 100 | 150 | 50 |
| **Cantidad de Iteraciones** | 1000 | 1000 | 500 | 100 | 10 | 10 | 10 |
| **Llega a solución** | NO | NO | NO | NO | SI | SI | SI |

En términos generales, y de acuerdo a las pruebas efectuadas, el algoritmo realiza un promedio de 242 iteraciones hasta converger en una solución. Y el tiempo de respuesta en promedio alcanza los 212segundos, es decir: 3 minutos 31 segundos.

1. Conclusión

Los métodos de algoritmos de enjambre pueden ser utilizados para resolver problemas criptoaritméticos complejos. Específicamente, el algoritmo firefly puede aplicarse para resolver de forma efectiva estos rompezabezas.

Como pueden observarse en las pruebas que se hicieron en la sección anterior, el número de soluciones posibles que presenta un puzzle en particular es el factor más determinante a la hora de evaluar el comportamiento de este algoritmo. Logrando mayor eficacia con problemas de múltiples soluciones.

Asimismo, existe un compromiso entre en número de luciérnagas, la cantidad de iteraciones finales y el tiempo de respuesta de la aplicación.

Además, la variación de la cantidad de iteraciones como configuración de entrada modifica el comportamiento del algoritmo de tal manera que puede hacer que no converja en una solución si es un número muy elevado. Esto se debe a que la ejecución del algoritmo cae en máximos locales. Para salir de ese estancamiento, recomendamos usar 10 iteraciones.

Finalmente, como el tiempo de respuesta es superior a los 3 minutos, a futuro se puede agregar restricciones referidas a los problemas específicos a resolver.

Referencias

1. Yang, X.-S.: Firefly algorithms for multimodal optimization, in: Stochastic Algorithms: Foundations and Applications, Lecture Notes in Computer Sciences, Vol. 5792, pp. 169-178 - SAGA (2009)
2. Mounier M., Aguirre F., Barboza M.: Resolución de Problemas Criptoaritméticos Utilizando Algoritmos Genéticos 43 JAIIO - EST 2014 - ISSN: 1850-2946 17º Concurso de Trabajos Estudiantiles (2014)
3. Ishaque A. S. Md., BahlulHaider Md., Wasid M. Al M., Alaul S. M., Kamrul Hassan Md., Ahsan T., Alam M. Sh.: Evolutionary Algo-rithm to Solve Cryptarithmetic Problem Transactions on engineering, computing and technology VI – World Enformatika Society Decembre (2004)
4. Holland, John H.: Algoritmos genéticos. Investigación y Ciencia (1992)
5. Molina, S.; Piccoli, M. F.; Leguizamón, G.: Algoritmos de inteligencia de enjambres sobre GPU: una revisión exhaustiva. XX Congreso Argentino de Ciencias de la Computación - Buenos Aires (2014)
6. Saraei, M., Analouei, R., Mansouri, P.: Solving of Travel-ling SalesmanProblemusingFireflyAlgorithmwithGreedyApproach. CumhuriyetUniversityFaculty of Science. Science Journal (CSJ), Vol. 36, No: 6 Special Issue (2015)
7. Fateen, S-E. K., Bonilla-Petriciolet, A.: Intelligent Firefly Allgorithm for Global Optimization. Springer International Publishing Switzerland (2014)
8. Dorigo, M., Blum, C.: Ant colony optimization theory: A survey. Theoretical computer science 344.2 (2005)
9. De-Los-Cobos-Da-Silva, S., Gutierrez-Andrade, M. A., Rincón-García, E.-A., Lara-Velázquez, P., Aguilar-Cornejo, M.: Colonia de abejas artificiales y optimización por enjambre de partículas para la estimación de parámetros de regresión no lineal. Revista de Matemática: Teoría y Aplicaciones 21.1 (2014)
10. Yang, X. S, Xingshi, He.: Firefly algorithm: recent advances and applications. International Journal of Swarm Intelligence 1.1 (2013)
11. Yang, X. S.: Firefly Algorithm, Stochastic Test Functions and Design Optimisation’, Int. J. Bio-Inspired Computation, Vol. 2, No. 2, pp.78–84 (2010)