ISSN 1870-4069

Comparación de la convergencia de cuatro algoritmos evolutivos al explorar conjuntos de soluciones discretas y continuas

Ibis Azael Arenas-Marín, Anabel Martinez-Vargas, M.A. Cosío-León

Universidad Politécnica de Pachuca, Zempoala, Hgo., México
iazaelarenas@micorreo.upp.edu.mx, {anabel.martinez, ma.cosio.leon}@upp.edu.mx

Resumen. La computación evolutiva es un área de investigación dentro de las ciencias computacionales que se basa en la metáfora de la evolución natural. Sus algoritmos llamados evolutivos son herramientas que brindan soluciones a aquellos problemas donde los algoritmos exactos resultan insuficientes. En este trabajo se presenta un análisis de convergencia de las soluciones realizado a cuatro algoritmos evolutivos, Optimización Binaria por Cúmulo de Partículas (BPSO), Optimización Binaria por Cúmulo de Partículas con Mutación (BPSOm), Optimización Binaria por Cúmulo de Partículas Modificado (MBPSO) y Algoritmo Genético Binario (BGA). Se analiza la forma como las soluciones decodificadas por la función objetivo convergen hacia un espacio de soluciones adecuado. Las pruebas realizadas muestran que BPSOm con 10 partículas puede aportar mejores soluciones en comparación con BGA que considera 50 individuos. Lo anterior permite plantear la posibilidad de utilizar BPSOm como una herramienta aplicable al problema de optimización de ubicaciones de vehículos aéreos no tripulados para proveer comunicaciones en situaciones de desastre en la búsqueda de mejores resultados.

Palabras clave: BPSO, algoritmo genético, convergencia, representación binaria, codificación.

A Convergence Comparison Analysis on Four Evolutionary Algorithms Exploring Discrete and Continuous Solution Sets

Abstract. Evolutionary computing is a research area within computer science that bases on the metaphor of natural evolution. Its algorithms known as evolutionary algorithms are tools to resolve problems where exact algorithms offer solutions too expensive in terms of computational time. This work proposes a convergence analysis over four evolutionary algorithms, Binary Particle Swarm Optimization (BPSO), Binary Particle Swarm Optimization with Mutation (BPSOm), Binary Particle Swarm Optimization Modified (MBPSO) and Binary Genetic Algorithm (BGA). The analysis covers the way they move to identify a subset of solutions of interest from the solution space. The results of simulations

show that BPSOm having 10 particles finds better solutions than BGA having 50 individuals. This allows us to consider the possibility of using BPSOm as a tool applicable to the problem of optimizing the locations of unmanned aerial vehicles to provide communications in disaster situations, to search for better results.

Keywords: BPSO, genetic algorithm, convergence, binary representation, coding.

1. Introducción

La selección de la representación de la solución para algoritmos evolutivos no es una tarea trivial. Una correcta elección de la representación de la solución coadyuvará al buen desempeño del algoritmo, mientras que una elección "erróneaçomprometería la eficiencia del algoritmo, ya que generará soluciones inadecuadas o explorará áreas indeseables del espacio de soluciones [1, 9].

Durante décadas se ha estudiado y desarrollado distintos tipos de representaciones que pueden emplearse en las diferentes clases de algoritmos evolutivos. Dentro de esas representaciones, se encuentran las de tipo binario y las de punto flotante. En general ambas representaciones poseen la capacidad de proporcionar soluciones adecuadas, sin embargo, la propia naturaleza de cada algoritmo provoca que estas representaciones tiendan a converger hacia óptimos locales o bien converjan hacia áreas de interés muy lentamente [1, 4].

Distinta literatura concuerda que las representaciones binarias de datos flotantes para algoritmos genéticos aportan soluciones con una mayor precisión, permitiendo aproximarlas a un espacio de soluciones deseable. Por ende la representación binaria por su simplicidad y practicidad se ha convertido en la más extendida en muchos algoritmos genéticos independientemente del problema que se está intentando resolver [1]

Otro algoritmo que hace uso de estas representaciones binarias es Binary Particle Swarm Optimization (por su sigla en Inglés, BPSO) [7]. Este, pertenece al grupo de los algoritmos evolutivos, su origen es el algoritmo Particle Swarm Optimization (por su sigla en inglés, PSO) [6], especializado en explorar espaciones de soluciones de tipo flotante, mientras que BPSO hace lo suyo en espacios de soluciones discretas. BPSO demostró que la representación binaria en algoritmos cuya naturaleza es la decodificación flotante, también puede aportar soluciones adecuadas [7]. Variantes del BPSO son el BPSO con una mutación (BPSOm) [8] y el Modified Binary Particle Swarm Optimization (MBPSO) [8], las cuales buscan evitar una problemática del BPSO que se identifica como saturación de la función sigmoide [8].

Cabe hacer mención que no toda la responsabilidad del desempeño de un algoritmo recae directamente en el tipo de representación que se utiliza, si bien es importante, también juegan un papel interesante en la búsqueda de las soluciones idóneas el proceso de selección de dichas soluciones y las distintas operaciones que se realizan con estas antes de ser evaluadas en una función objetivo.

En este trabajo de investigación analizamos la convergencia en el uso de la representación binaria utilizada en [9] para números de punto flotante y números enteros en: Algoritmo Genético Binario (BGA), BPSO, BPSOm y MBPSO. Para describir el proceso investigativo y los resultados obtenidos, este articulo está organizado de la siguiente manera: la Sección 2 detalla las características de cada uno de los algoritmos evolutivos analizados. Las secciones 3 y 4 muestran los parámetros de simulación así como las observaciones realizadas durante las ejecuciones de cada uno de los algoritmos. Finalmente la Sección 5 concluye el trabajo y menciona el trabajo a futuro.

2. Algoritmos evolutivos

Un algoritmo evolutivo cuenta con tres procesos principales, inicialización, donde intervienen los distintos tipos de representación de las soluciones; evaluación, en donde las soluciones son decodificadas y puestas a prueba y finalmente la supervivencia, donde una solución puede permanecer sin cambio o bien aportar a la siguiente generación [5]. Tanto en Algoritmos Genéticos como en Cúmulo de Partículas estos comportamientos generales son observables, aunque no del mismo modo en cada uno de ellos, bastaría citar el cruzamiento (descrito en 2.1) y la memoria de BPSO como elementos característicos de cada algoritmo lo cual los vuelve diferentes. Vistos como herramientas que buscan explorar espacios de soluciones en busca de soluciones óptimas, o bien ubicando áreas de soluciones deseables, los algoritmos evolutivos deben apoyarse de ciertas prácticas que mejoran su eficiencia.

Desde su aparición una de las principales áreas de estudio en algoritmos evolutivos ha sido la adecuada selección y uso de representaciones para las soluciones. Diversos estudios refieren que el uso de representaciones más cercanas a la naturaleza del problema son las que mejores resultados aportan [9]. Esto no representa una regla, ya que por otro lado varios autores documentan que las características propias del algoritmo son también limitantes. En ocasiones estas resultan en desventajas durante la búsqueda de un área de soluciones idónea [2,4,9], lo cual abre una puerta al análisis de las distintas representaciones en los diferentes tipos de algoritmos evolutivos.

Durante décadas las representaciones de tipo binaria y las de tipo flotante se han extendido como las más utilizadas dentro del área de algoritmos evolutivos. En este documento se evalúa la representación binaria como una herramienta que permita una convergencia aceptable a un espacio de soluciones idóneo en cada uno de los algoritmos analizados.

2.1. Algoritmo genético binario

Un BGA, es un caso particular de los algoritmos evolutivos que utiliza técnicas inspiradas en la naturaleza para simular la evolución natural [1,3,4]. Desde su aparición en 1975 ha sido el algoritmo evolutivo de más extenso uso. Básicamente se trata de un grupo de individuos cuya aptitud determina su reproducción y

supervivencia [1, 4]. Aunque al igual que en la naturaleza no siempre sobreviven los más aptos, en el BGA las nuevas generaciones de individuos podrían contener individuos con aptitudes no muy buenas. Sin embargo, esta es una característica de estos algoritmos, misma que le permite explorar de forma adecuada el espacio de soluciones [4, 9]. Una descripción detallada de las características de los algoritmos genéticos puede encontrarse en [1, 3, 4, 9]

En los algoritmos genéticos la representación de las soluciones es la abstracción de los cromosomas de un individuo. Esta representación determina alguna característica del problema a resolver e influye directamente en los tipos de operadores genéticos que se usarán. Para el BGA se trata de una representación binaria, la cual consiste en generar cadenas de longitud n, de forma binaria (1 ó 0). Cada una de las cadenas representará un cromosoma, mientras que cada uno de los elementos (1 ó 0) representará un gen de dicho cromosoma [1–3] y la decodificación de este cromosoma será el fenotipo.

La forma más extendida de inicialización de los algoritmos evolutivos es la generación aleatoria de cadenas binarias, donde cada solución tiene la misma probabilidad de ser ubicada dentro del espacio de soluciones.

Generalmente para detener un algoritmo genético se utiliza un criterio de parada basado en alcanzar un determinado número de generaciones. Criterios de parada diferentes son expuestos en [1, 4, 9]. Para la evaluación de los algoritmos genéticos la función objetivo no afecta directamente en su comportamiento interno, es decir, el comportamiento del algoritmo es independiente de la función de evaluación propuesta [4]

La selección dentro de los algoritmos genéticos está determinada por la aptitud de cada individuo. Si se trata de un problema de maximización, con una aptitud alta, el individuo tendrá mayores probabilidades de sobrevivir a la siguiente generación y heredar parte de sus características a los nuevos individuos. Desde la aparición de los Algoritmos Genéticos (GA), los procesos de selección de individuos y/o padres han sido estudiados y modificados [3, 9]. La forma más utilizada y que es usada en este trabajo, es el método de ruleta detallado en [1, 3]. Para otros métodos de selección puede consultarse [1, 3].

En algoritmos genéticos con representación binaria, el cruce consiste en dada una probabilidad de cruce, se generará un número aleatorio entre rango de $(0, 1)$. Si este número es menor a la probabilidad de cruce entonces se realizará el cruce entre las parejas de los individuos seleccionados. Para determinar cuál será la información genética que intercambiaran los padres se hace uso de cortes mediante la generación de números aleatorios entre $[1, n - 1]$ donde n es la longitud del cromosoma. En este trabajo se utilizará el cruce por un solo punto. Por ello se utiliza la expresión $(n - 1)$, ya que un corte en el punto n daría el mismo cruce que un corte en el punto 1, es decir intercambiarían la cadena completa de genes.

En la naturaleza la mutación es una peculiaridad de cada individuo que sucede no muy cotidianamente. En los GA una mutación está determinada por una probabilidad de mutación en cada uno de los genes. Al igual que en los procesos anteriores se presenta mediante la generación de números aleatorios

entre $(0, 1)$ donde, si el número generado de manera aleatoria es menor a la probabilidad de mutación se presentará la mutación en el gen en turno, de lo contrario el gen permanecerá sin alteraciones. La mutación permite a las soluciones candidatas (individuos) cambiar su aptitud, sin embargo, no siempre esta mutación aumentará la aptitud de los individuos. Por ejemplo, una cadena de 8 bits, donde el bit más significativo sea el que se verá afectado por la mutación, donde la función de evaluación está determinada por $f(x) = x^2$ y se trate de un problema de maximización, el genotipo **[1 0 0 0 0 0 0 0 1]** representaría el fenotipo 129 y produciría el fitness de 16641, sin embargo al sufrir una mutación en el bit más significativo el genotipo resultaría **[0 0 0 0 0 0 0 1]** el cual representa el fenotipo 1 que evaluado en la función objetivo daría como resultado 1, lo cual lo convierte en una solución indeseable por su pobre desempeño.

2.2. Algoritmo BPSO y variantes

BPSO vio su primera aparición en el mundo de los algoritmos evolutivos en 1997 como una opción de PSO [7] con representación binaria. Es un algoritmo cuya naturaleza es buscar soluciones mediante el análisis de cadenas binarias, que con auxilio de parámetros de constricción y de interacción social proveen de soluciones a los problemas de programación no lineal [7, 8]. En BPSO, un conjunto de partículas es evaluado en una función objetivo una a una y el desempeño de cada una de estas partículas es almacenado en una memoria. A diferencia de GA, BPSO cuenta con la cualidad de almacenar por cada iteración la mejor partícula (solución candidata). En resumen, este algoritmo cuenta con dos funciones básicas: la actualización de la velocidad y la actualización de la posición. Sin embargo presenta cierta tendencia a la saturación de las función sigmoide [8].

En BPSO la forma de inicializar las partículas es similar a la utilizada en BGA, se inicializan la matriz de posición y velocidad de forma aleatoria y la matriz de memoria en su estado inicial será una copia de la matriz de posición [1, 7]. En BPSO el criterio de parada más utilizado es alcanzar un determinado número de iteraciones para después obtener la mejor partícula almacenada en la memoria que será la mejor solución encontrada por el algoritmo para el problema propuesto.

Las operaciones de actualización de velocidad (Ecuación (1)) y posición (Ecuación (2) y Ecuación (3)) en BPSO están determinadas por:

$$v_{(i,j)}(t+1) = wv_{(i,j)}(t) + c_1R_1(p_{best,i,j} - x_{i,j}(t)) + c_2R_2(g_{best,i,j} - x_{i,j}(t)), \quad (1)$$

$$x_{i,j}(t+1) = \begin{cases} 0 \text{ si rand}(\ ) \geq S(v_{i,j}(t+1)), \\ 1 \text{ si rand}(\ ) < S(v_{i,j}(t+1)), \end{cases} \quad (2)$$

$$S(v_{i,j}(t+1)) = 1/(1 + e^{(-v_{i,j}(t+1)}), \quad (3)$$

donde: $v_{i,j}(t)$ representa la velocidad de la *n-ésima* partícula, $x_{i,j}(t)$ representa la posición de la partícula, w es el coeficiente de inercia, c1 y c2 son los coeficientes

cognitivo y social, R1 y R2 son números aleatorios entre 0 y 1, $p_{best,i,j}$ es la mejor posición de la partícula, y $g_{best,i,j}$ es la mejor posición del cúmulo para Ecuación (1). $x_{i,j}(t+1)$ representa la posición de la i-ésima partícula, rand() es un numero aleatorio entre (0,1) y $S(v_{i,j}(t+1))$ representa el valor de la funcion sigmoide evaluada mediante el valor de $v_{i,j}(t+1)$. En cada una de las iteraciones, la velocidad modifica la posición de la partícula de acuerdo a las ecuaciones expuestas en [9] por lo que se puede considerar un algoritmo más simple que el algoritmo genético, más no por esto resulta ser un algoritmo menos completo.

BPSO con una mutación Lee *et al.* en [8] hace notar cierta tendencia de BPSO a la saturación de la función sigmoide, lo cual haría que las soluciones localizadas por BPSO tiendan a converger de forma prematura a un óptimo local, y que se vean comprometidas para lograr salir de este. Para ello propone una mutación mediante:

$$\begin{aligned} &Para(i=1; i<n; i=i+1)\{ \\ &if(rand() < r_{mu}) \\ &then v_{i,jr}(t+1) = -v_{i,jr}(t+1)\}, \end{aligned} \tag{4}$$

donde r_{mu} es un la probabilidad de mutación, $v_{i,jr}$ es el elemento dentro de la matriz de velocidad en la ubicación (i, jr), la cual permitirá cambiar el sentido cuando los valores de la velocidad están muy cercanos a la velocidad máxima o la velocidad mínima y así evitar una convergencia prematura. La ecuaciónes (1) (2) y (3) permanecen sin cambios por lo que el pseudocódigo quedaría de la siguiente manera:

Algoritmo 1 BPSOm

```
1: INICIO
2: inicializar particulas x_{i,j}(t)
3: evaluar x_{i,j}(t)
4: mientras condicion_de_parada ≠ cierto hacer
5:    v_{i,j}(t) ← actualizar v_{i,j}(t) {mediante (1)}
6:    v_{i,j}(t) ← actualizar v_{i,j}(t) {por medio de (4)}
7:    x_{i,j}(t) ← actualizar x_{i,j}(t) {por medio de (2) y (3)}
8:    evaluar x_{i,j}(t)
9:    t=t+1
10: fin mientras
11: FIN
```

MBPSO. En MBPSO el concepto de genotipo-fenotipo es aplicado a BPSO de la siguiente manera: considere la velocidad de BPSO como un genotipo y la posición binaria de BPSO como un fenotipo. Luego las operaciones de actualizacion de velocidad y posicion (1), (2) (3) de BPSO original y la operación de mutación (4) de BPSOm, cambian por las siguientes expresiones:

$$v_{(i,j)}(t+1) = wv_{(i,j)}(t) + c_1R_1(p_{best,i,j} - x_{i,j}(t)) + c_2R_2(g_{best,i,j} - x_{i,j}(t)), \tag{5}$$

$$x_{g,i,j}(t+1) = x_{g,i,j}(t) + v_{,i,j}(t+1), \tag{6}$$

$$\begin{array}{l} para(i=1; i<n; i=i+1) \\ if(rand() < r_{mu}) \\ then x_{g,i,jr}(t+1) = -x_{g,i,jr}(t+1), \end{array} \tag{7}$$

$$x_{p,i,j}(t+1) = \begin{cases} 0 \text{ si rand( )} \geq S(x_{g,i,j}(t+1)), \\ 1 \text{ si rand( )} < S(x_{g,i,j}(t+1)), \end{cases} \tag{8}$$

$$S(x_{g,i,j}(t+1)) = 1/(1+e^{(-x_{g,i,j}(t+1)}), \tag{9}$$

donde: $v_{i,j}(t)$ representa la velocidad de la n-ésima partícula, $x_{i,j}(t)$ representa la posición de la partícula, w es el coeficiente de inercia, $c1$ y $c2$ son los coeficientes cognitivo y social, $R1$ y $R2$ son números aleatorios ente 0 y 1, $p_{best,i,j}$ es la mejor posición de la partícula actual, $g_{best,i,j}$ es la mejor posición del cúmulo. No confundir $x_{g,i,j}$ con $g_{best,i,j}$; $x_{g,i,j}$ representa la matriz del genotipo calculada a través de (6). $x_{p,i,j}$ representa al fenotipo obtenido por medio de (7) (8) y (9), no confundir con $p_{best,i,j}$. El pseudocódigo es:

Algoritmo 2 MBPSO

```
1: INICIO
2: inicializar particulas x_{p,i,j}(t)
3: evaluar x_{p,i,j}(t)
4: mientras condicion_de_parada ≠ cierto hacer
5:    v_{i,j}(t) ← actualizar v_{i,j}(t) {mediante (5)}
6:    x_{g,i,j}(t) ← actualizar x_{g,i,j}(t) {mediante (6)}
7:    x_{g,i,j}(t) ← mutar x_{g,i,j}(t) {por medio de (7)}
8:    x_{p,i,j}(t) ← actualizar x_{p,i,j}(t) {por medio de (8) y (9)}
9:    evaluar x_{p,i,j}(t)
10:   t=t+1
11: fin mientras
12: FIN
```

3. Simulación

Para las pruebas de optimización se utilizó la función uni-modal siguiente:

$$\begin{array}{l} f(x) = \frac{sin(\pi * x)}{256}, \\ 0 \leq x \leq 255, \end{array} \tag{10}$$

y cuya gráfica se muestra en la Fig. 1.

Como se puede observar es una función que permite evaluar las convergencias de las soluciones de los algoritmos antes mencionados, encontrando un máximo

global en $x = 128$.Tiene áreas de interés entre los valores de x de 125 y 137 como puede observarse en el área verde de la Fig. 1.

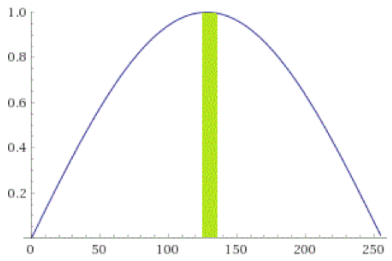


Fig. 1. Gráfica de la función objetivo.

Además, presenta un reto durante la exploración al tener la dificultad de cambiar su valor de $x = 127$ a 128 al utilizar la codificación binaria, por la cantidad de bits que debe cambiarse.

Tal como se describe en las secciones anteriores la forma de representar las soluciones de estos algoritmos está determinada por cadenas binarias, auxiliándonos de las ecuaciones (11) y (12) para la codificación y decodificación de números de punto flotante, el proceso completo se encuentra descrito en [4]:

$$a_i \leq (b_j - a_j) * (10)^n \leq 2^{mj} - 1, \tag{11}$$

$$x_j = a_j + decimal(substring) \times \frac{(b_j - a_j)}{(2^{mj} - 1)}, \tag{12}$$

donde a_i representa el valor mínimo del área de exploración, b_j es el valor máximo del área de exploración, n la cantidad de dígitos después del punto decimal y m_j es la longitud de la cadena.

La ejecución de los algoritmos se realizó en un equipo de cómputo de las siguientes características Core2Duo a 2.1 GHz con 4 GB de RAM DDR3 a 1333 MHz, y un HDD de 320 GB a 5200 RPM. El área de exploración de los algoritmos estuvo limitada a los valores de [0, 255], con un óptimo global en $x = 128$. Los parámetros utilizados para cada algoritmo genético BGA se encuentran listados en la Tabla 1 y para los BPSO y variantes en la Tabla 2.

Tabla 1. Parámetros utilizados en BGA con decodificación flotante y entera.

Parámetro	BGA decodificación Flotante	BGA decodificación Entera
Porcentaje de cruce	0.25	0.25
Porcentaje de mutación	0.1	0.1
Longitud de la cadena de bits	22	8
Cantidad de iteraciones	1000	1000
Dígitos después del punto	3	3

Tabla 2. Parámetros utilizados para BPSO decodificación entera y flotante.

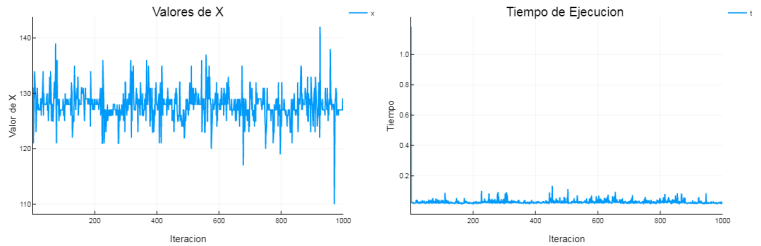
Parámetro	decodificación Flotante			decodificación Entera		
	BPSO	BPSOm	MBPSO	BPSO	BPSOm	MBPSO
$[-V_{min}, V_{max}]$	[-4.0,4.0]					
Longitud de la cadena de Bits	22			8		
c1 y c2	1.42694					
w	0.68343					
Probabilidad de mutación	N/A	0.3	0.3	N/A	0.3	0.3
Cantidad de iteraciones	1000					
Digitos despúes del punto	3			N/A		

Cada uno de los algoritmos fue programado en el lenguaje programación **Julia 6.2.2**, bajo el entorno de desarrollo Atom.

4. Resultados

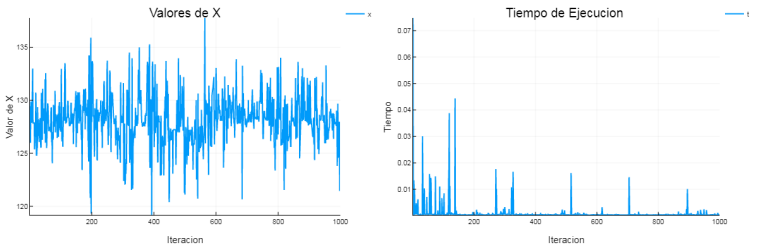
Se realizaron 30 ejecuciones de cada uno de los algoritmos. Las gráficas en Figura 2(a) y Figura 3(a) pertenecen a las mejores ejecuciones y reportan los valores generados en cada iteración del BGA con codificación flotante y BGA con codificación entera respectivamente. Las gráficas en Figura 2(b) y Figura 3(b) representan el tiempo (en segundos) de esas mejores ejecuciones para realizar las operaciones de evaluación, cruce, mutación y evolución en cada una de las iteraciones.

Para algoritmos genéticos es apreciable que la convergencia de las soluciones resulta demasiado lenta, y tal como lo definen algunos autores se puede observar que su característica mayormente marcada es la exploración del área de soluciones. Por otro lado, en cuestión de tiempo como se puede ver en las Figura 2(b) y Figura 3(b), permiten concluir que la representación binaria para codificación flotante con 3 dígitos después del punto decimal, en BGA converge de manera más lenta que la codificación entera.



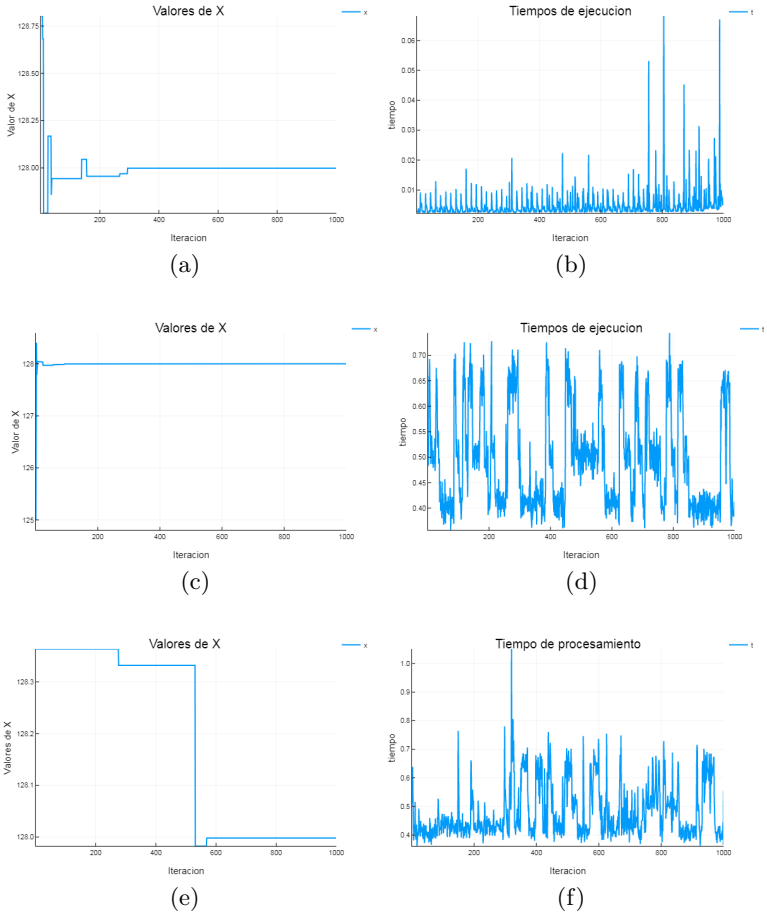
(a) (b)

Fig. 2. Codificación entera de BGA para 50 individuos con 8 bits de longitud.



(a) (b)

Fig. 3. Codificación flotante de BGA para 50 individuos con 22 bits de longitud.



(a) (b) (c) (d) (e) (f)

Fig. 4. Codificación flotante para 50 partículas de los algoritmos: (a) BPSO; (c) BPSOm; y, (e) MBPSO.

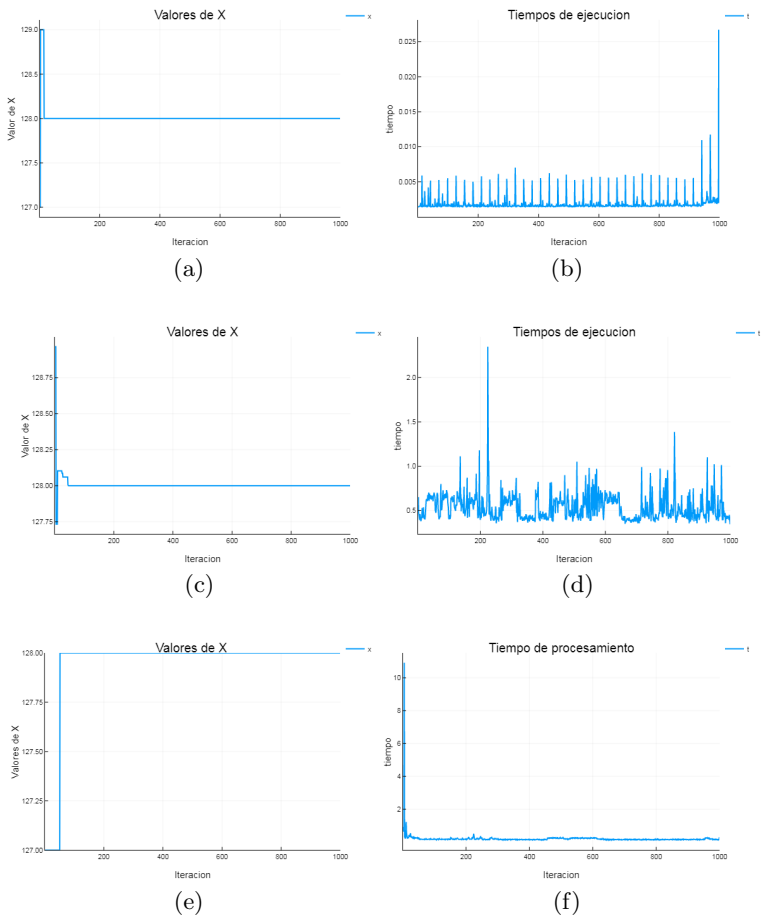


Fig. 5. Codificación entera para 50 partículas de los algoritmos: (a) BPSO; (c) BPSOm; y, (e) MBPSO.

Las gráficas en Figura 4 muestran las mejores ejecuciones para BPSO, BPSOm, y MBPSO con codificación flotante respectivamente. Las gráficas en Figura 4(a), Figura 4(c) y Figura 4(e) muestran las mejores ejecuciones para BPSO, BPSOm y MBPSO con codificación flotante respectivamente.

Las gráficas en Figura 4(b), Figura 4(d) y Figura 4(f) representan el tiempo de las mejores ejecuciones.

Las gráficas en Figura 5(a), Figura 5(c) y Figura 5(e) muestran las mejores ejecuciones para BPSO, BPSOm y MBPSO con codificación entera. Las gráficas en Figura 5(b), Figura 5(d) y Figura 5(f) representan el tiempo de las mejores ejecuciones respectivamente.

Las tablas 3 y 4 muestran las mediciones realizadas a 30 ejecuciones para cada algoritmo. Como puede apreciarse BPSOm y MBPSO generan resultados con mayor precisión que BGA. Además cabe resaltar la capacidad que tiene BPSOm para realizar una mejor aproximación al óptimo global con una cantidad de recursos mínimos. Se hace también perceptible el desempeño promedio de cada algoritmo, observando que MBPSO resulta ser el algoritmo con mejor desempeño promedio.

Tabla 3. Calidad de las soluciones para codificación entera.

Partículas /Individuos	Algoritmo	Mejor aptitud	Peor aptitud	Mejor valor de x	Peor valor de x	Aptitud promedio	Valor de x promedio	Tiempo promedio de ejecución (s)
10	BGA	1	0.63439328416	128	56.0	0.98842140599	127.518933	0.1296936203
	BPSO	1	0.91911385169	128	95.0	0.99995985537	127.518933	0.1296936203
	BPSOm	1	0.89867446569	128	91.0	0.99997897453	127.518933	0.1298182704
	MBPSO	1	0.99729045667	128	122.0	0.99999560840	128.002333	30.579896232
30	BGA	1	0.96377606579	128	126.0	0.99886441814	128.123666	0.2136438140
	BPSO	1	0.99992470183	128	127	0.99999977410	127.999	0.7063252750
	BPSOm	1	0.99879545620	128	124.0	0.99999761568	127.983	81.428241680
	MBPSO	1	0.99932238458	128	131.0	0.99999859454	128.00466	87.903250711
50	BGA	1	0.99090263542	128	139.0	0.99961292221	128.0083333	0.4128423363
	BPSO	1	0.99992470183	128	129.0	0.99999974900	128.0013333	1.6418504633
	BPSOm	1	0.99247953459	128	118.0	0.99999842451	127.99926666	137.75230444
	MBPSO	1	0.99932238458	128	131.0	0.99999957333	128.003	153.73698562

Tabla 4. Calidad de las soluciones para codificación de punto flotante.

Partículas /Individuos	Algoritmo	Mejor aptitud	Peor aptitud	Mejor valor de x	Peor valor de x	Aptitud promedio	Valor de x promedio	Tiempo promedio de ejecución (s)
10	BGA	0.999999999998	0.5791642999	127.9998404	205.6650163	0.9884305	127.8960934	0.1281748
	BPSO	0.999999999999	0.740624959	127.9999620	67.9604334	0.9999446	127.9989473	0.1561739
	BPSOm	0.999999999999	0.8043359628	128.0000228	76.15455893	0.9999592	127.9754834	69.554611
	MBPSO	0.999999999998	0.9822086929	128.0001444	143.3941253	0.9999919	127.9910014	63.779639
30	BGA	0.999999999999	0.8350242862	128.0000228	80.52389991	0.9969653	127.8809693	0.1715704
	BPSO	0.999999999998	0.9884961460	128.0001444	115.6278838	0.9999973	127.9991538	1.0581364
	BPSOm	0.999999999999	0.9868285867	128.0000228	141.2403395	0.9999959	127.9854557	219.63925
	MBPSO	0.999999999999	0.9953752558	128.0000228	135.840007	0.9999989	128.0004667	184.40917
50	BGA	0.999999999999	0.9740595627	128.0000228	146.6010324	0.9995684	127.9390172	0.4003717
	BPSO	0.999999999999	0.9970689913	127.9390172	134.2405055	0.9999988	128.0009653	2.9002064
	BPSOm	0.9999999999999	0.9931025973	128.0000228	137.5763017	0.9999978	127.9864345	359.92917
	MBPSO	0.9999999999999	0.9992821536	128.0000228	124.9122166	0.9999995	127.9970819	398.89039

Considerando la información de las tablas, la convergencia observada en los algoritmos de cúmulo de partículas resulta interesante al observar que en las evaluaciones de 10 partículas, son capaces de alcanzar una buena aproximación. En algunos casos mejor a la de GA, sin consumir mayores recursos (poblaciones grandes) lo cual los vuelve candidatos a ser utilizados en la búsqueda de soluciones, cuando los recursos y/o el tiempo es una variable a considerar.

Se debe hacer mención que la propiedad de memoria de cúmulo de partículas, le permite tener un parámetro de mejora (experiencia) y es esto lo que se puede apreciar en las gráficas de convergencia. Una vez que BPSO ha encontrado una solución suficientemente buena, la probabilidad de que esta cambie está determinada por la exploración.

5. Conclusión y trabajo futuro

La exploración es mejorada por las variantes de BPSOm y MBPSO. Estas evitan un atrapamiento en óptimos locales con convergencia rápida y mejoran la exploración, sin alterar la capacidad de aprendizaje de BPSO.

Durante las pruebas se pudo apreciar la capacidad que tiene BPSO y sus variantes sobre BGA para localizar de forma rápida un área de soluciones adecuada, aun cuando sus recursos son limitados. Si bien el tiempo es un parámetro interesante donde BGA supera a BPSO y sus variantes. Se puede visualizar que BPSOm con 10 partículas logra una aptitud semejante a BGA con 50 individuos y, en promedio, su fitness resulta mayor. Por lo que se puede considerar el tiempo como una variable de poco impacto en el análisis de la convergencia.

En general BPSOm registra mejores valores tanto para aptitud como para valores de x que cualquiera de los otros algoritmos analizados. Como trabajo futuro, se implementarán las variantes de BPSO presentadas en esta investigación, para ubicación de vehículos aéreos no tripulados en zonas de desastre. Además, se cambiará el enfoque empírico hacia un análisis cuantitativo del comportamiento de cada uno de los algoritmos.

Agradecimientos. Trabajo apoyado por CONACyT mediante beca nacional.

Referencias

1. Eiben, A.E., Smith, J.E.: Theory. In: Introduction to Evolutionary Computing. Springer (Jan 2015)
2. Goldberg, D.E.: Genetic algorithms. Pearson Education India (2006)
3. Holland, J.H.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press (1992)
4. Houck, C.R., Joines, J., Kay, M.G.: A genetic algorithm for function optimization: a Matlab implementation. Ncsu-ie tr 95(09), 1–10 (1995)
5. Kachitvichyanukul, V.: Comparison of three evolutionary algorithms: GA, PSO, and DE. Industrial Engineering and Management Systems 11(3), 215–223 (2012)
6. Kennedy, J.: Particle swarm optimization. In: Encyclopedia of machine learning, pp. 760–766. Springer (2011)
7. Kennedy, J., Eberhart, R.C.: A discrete binary version of the particle swarm algorithm. In: Systems, Man, and Cybernetics, Computational Cybernetics and Simulation, IEEE International Conference on. vol. 5, pp. 4104–4108. IEEE (1997)
8. Lee, S., Soak, S., Oh, S., Pedrycz, W., Jeon, M.: Modified binary particle swarm optimization. Progress in Natural Science 18(9), 1161–1166 (2008)
9. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution. Springer: New York (1992)