

Capítulo

4

El algoritmo de optimización por enjambre de partículas (PSO)

Objetivos

El objetivo de este capítulo es presentar la analogía y los conceptos básicos del algoritmo de optimización por enjambre de partículas ("Particle Swarm Optimization", PSO). Con esto, el lector será capaz de aplicarlo en implementaciones propias. De forma general, se describe la teoría referente a PSO y su codificación. Por medio de un ejemplo práctico se analizan los operadores principales y sus aplicaciones, mientras que de forma teórica se estudian algunas modificaciones propuestas en la literatura.

4.1 Introducción a la optimización por enjambre de partículas

Desde sus inicios la optimización por enjambre de partículas ha atraído la atención de los investigadores en diversas áreas de ingeniería y ciencias. Existen diferentes campos de estudio en los cuales hay situaciones que exigen encontrar soluciones que puedan considerarse como mejores, con base en ciertos criterios establecidos. Algunos de esos problemas no permiten el uso de métodos tradicionales de optimización, porque además de realizar operaciones complejas, pueden quedar atrapados en soluciones locales. Para hacer frente a estas situaciones, el algoritmo PSO es fácil de implementar y sus operadores son sencillos de comprender para el lector. Además, por medio de la distribución de la población de partículas en el espacio de búsqueda y de los operadores propios de PSO se evita caer en los óptimos locales y con esto dar soluciones imprecisas.

La optimización por enjambre de partículas estándar fue originalmente propuesta por Kennedy y Eberhart^[31] y a pesar de que existe una gran cantidad de modificaciones e hibridaciones, este capítulo se centra en el estudio del método PSO con sus operadores básicos. Además de esto también se analizarán algunas de las modificaciones de PSO que son más aceptadas en la comunidad científica y su aplicación a problemas con restricciones. Actualmente, es posible encontrar en la literatura referente aplicaciones que involucran el uso de enjambres de partículas en áreas como: el control automático^[32], el diseño de rutas de vehículos^[33], planeación de rutas para robots^[34], estimación del consumo eléctrico^[35], etcétera. Con lo que se comprueba que PSO sigue siendo una alternativa confiable a las técnicas tradicionales.

A partir del origen del PSO ha surgido una nueva vertiente en los algoritmos metaheurísticos denominada: inteligencia de enjambre^[36]. Una gran cantidad de técnicas metaheurísticas usan la inteligencia de enjambre como parte de sus operadores. Aunque este concepto se puede aplicar con diferentes analogías de la naturaleza, la propuesta original del PSO está inspirada en el comportamiento que tienen las bandadas de aves o los bancos de peces^[31],^[2]. De forma general, para este algoritmo se tiene una población de partículas, las cuales operan sobre un espacio de búsqueda acotado. Para cada iteración se generan nuevas posiciones para las partículas, las cuales son obtenidas usando una velocidad que se calcula considerando la mejor posición global y la mejor posición actual de cada partícula. Para determinar la calidad de población se requiere de una función objetiva, donde se evalúan los individuos cada vez que toman una nueva posición. En la selección de los miembros de la población se debe saber si se trata de un problema de maximización o minimización, de esta forma los operadores de PSO serán empleados correctamente. En cada iteración los elementos de la población comparten información, la cual les permite acercarse con mayor o menor rapidez a la solución global del problema.

A diferencia de otros métodos similares, la optimización por enjambre de partículas no tiene operaciones que implican el cruzamiento o la mutación de elementos de la población. Con esto se evita el uso de sistemas de numeración distintos al del espacio de búsqueda. Los operadores de PSO permiten explorar el espacio de búsqueda y a la vez explotar las regiones en las que se tiene más probabilidad de encontrar el óptimo global. Este capítulo describe la teoría referente a PSO y además de explicar los operadores del algoritmo, también se explican algunas versiones de PSO modificadas para incrementar su rendimiento en diversos problemas de optimización. Finalmente se muestra un ejemplo práctico codificado usando el algoritmo PSO estándar, para que el lector pueda aplicarlo fácilmente.

4.2 Optimización por enjambre de partículas

El modelo básico del PSO considera las soluciones candidatas como partículas, las cuales de forma inicial están distribuidas de manera aleatoria en un espacio de búsqueda, este conjunto de partículas es conocido como la población inicial. La calidad de la población se determina al evaluar a los individuos en la función objetivo, con esto también se determina al mejor de los elementos. Las nuevas posiciones de las partículas y la velocidad con que éstas son desplazadas, se calculan considerando el valor del mejor elemento global y el valor actual de cada partícula en conjunto con un número aleatorio. Cada vez que las partículas son desplazadas, deben de evaluarse nuevamente en la función objetivo. Solamente se actualizan las mejores partículas actual y global, cuando se encuentran valores que mejoras a los que se tenían almacenados. El pseudocódigo del algoritmo 4.1 muestra las fases principales del algoritmo PSO estándar, mientras que en las secciones siguientes se detalla cada una de estas fases.

Algoritmo 4.1 Optimización por enjambre de partículas	
1.	Inicializar la población de las partículas
2.	Evaluar la población en la función objetivo y elegir la mejor partícula
3.	while (criterio de paro)
4.	for (todas las partículas en todas las dimensiones)
5.	Generar una nueva velocidad
6.	Calcular una nueva posición
7.	Evaluar la función objetivo y elegir la mejor partícula
8.	end for
9.	Actualizar la mejor partícula de la población
10.	end while

4.2.1 Inicialización >

Como en la mayoría de los algoritmos metaheurísticos PSO requiere de una etapa de inicialización. De forma genérica en esta fase se crean las soluciones candidatas que a través del proceso de optimización serán modificadas por los operadores propios de cada algoritmo. En el caso del PSO, la población inicial de partículas (soluciones candidatas), se genera distribuyendo aleatoriamente las soluciones en un espacio de búsqueda acotado por los límites superior e inferior definidos previamente. En esta fase también se definen los parámetros del problema a optimizar, es decir las dimensiones, límites del espacio de búsqueda y las restricciones (en caso de que existan). La ecuación 4.1 describe la optimización de las partículas para PSO.

$$x_k^{i,t} = l_k + rand(u_k - l_k), x_k^{i,t} \in \mathbf{x}^t \quad (4.1)$$

Donde: $x_k^{i,t}$ es la i -ésima partícula de la población x^t , i es el índice que se refiere al número de partículas y tiene como valor máximo el tamaño de la población ($i = 1, 2, \dots, \text{Par}_N$). La dimensión del

problema se define por la variable k y el número de iteraciones con la variable t . Mientras l_k y u_k son los límites inferior y superior respectivamente para una de las dimensiones del espacio de búsqueda, por último $rand$ es un número aleatorio uniformemente distribuido entre el rango de cero a uno.

4.2.2 Velocidad de las partículas >

Para poder obtener la nueva posición que tendrán las partículas en el espacio de búsqueda, primero es necesario calcular la velocidad de cada una de ellas. Para esto se necesita el valor previo de la velocidad, si se trata de la primer iteración este valor será igual a cero. Además se necesitan los mejores valores globales y locales de cada partícula. En la ecuación 4.2 se presenta cómo se lleva a cabo el cálculo de dicha velocidad.

$$\mathbf{v}^{t+1} = \mathbf{v}^t + rand1 \times (\mathbf{P} - \mathbf{X}^t) + rand2 \times (\mathbf{G} - \mathbf{X}^t) \quad (4.2)$$

Donde: \mathbf{v}^{t+1} es el valor de la velocidad que se calcula para la iteración $t+1$, \mathbf{v}^t es la velocidad en la iteración anterior t , \mathbf{x}^t es el vector que contiene las posiciones de cada partícula, \mathbf{P} contiene las mejores posiciones actuales asociadas a la vecindad de cada partícula, mientras que \mathbf{G} es la mejor partícula actual a nivel global. Por otra parte, $rand1$ y $rand2$ son números aleatorios usualmente distribuidos de forma uniforme en el rango de cero a uno. Las operaciones matemáticas se realizan de forma vectorial respetando las reglas del álgebra de vectores. En la figura 4.1 se describen gráficamente los elementos que intervienen tanto en el cálculo de la velocidad, como en la obtención de las nuevas posiciones de cada partícula.

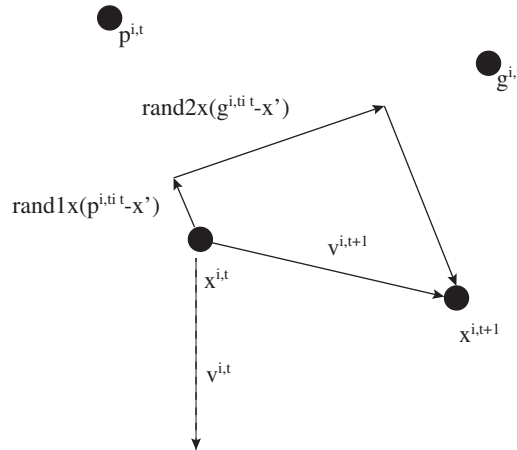


Figura 4.1. Descripción gráfica del cálculo de velocidades y movimiento de partículas en PSO.

Los valores aleatorios $rand1$ y $rand2$ tienen como propósito en la ecuación 4.2, dar la posibilidad de que se elija cualquier trayectoria siempre y cuando se siga la dirección del óptimo local y global. Ambas direcciones se conjuntan en una sola operación. Es importante mencionar al lector que la mayoría de las modificaciones del PSO que se pueden encontrar en la literatura, realizan alteraciones al cálculo de la velocidad. De forma tal que el autor tiene la libertad de proponer nuevas versiones del algoritmo.

4.2.3 Movimiento de las partículas >

Después de calcular la velocidad las partículas son desplazadas hacia nuevas posiciones en la iteración actual. Para realizar este movimiento se realiza una simple operación donde se combina la velocidad con las posiciones anteriores de la población, este operador se describe en la ecuación 4.3.

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \mathbf{v}^{t+1} \quad (4.3)$$

Donde \mathbf{x}^{t+1} es el vector donde son almacenadas las nuevas posiciones obtenidas en la iteración $t+1$, \mathbf{x}^t corresponde a las posiciones previas de las partículas, es decir, las que se calcularon en la iteración t . Finalmente \mathbf{v}^{t+1} es el vector de velocidad que se obtuvo usando la ecuación 4.2. En la literatura y como parte de las modificaciones, se pueden encontrar implementaciones que alteran la forma en que las partículas son desplazadas.

4.2.4 Estructura básica del algoritmo PSO >

En el diagrama de flujo de la figura 4.2 se muestra a detalle la estructura básica del algoritmo PSO.

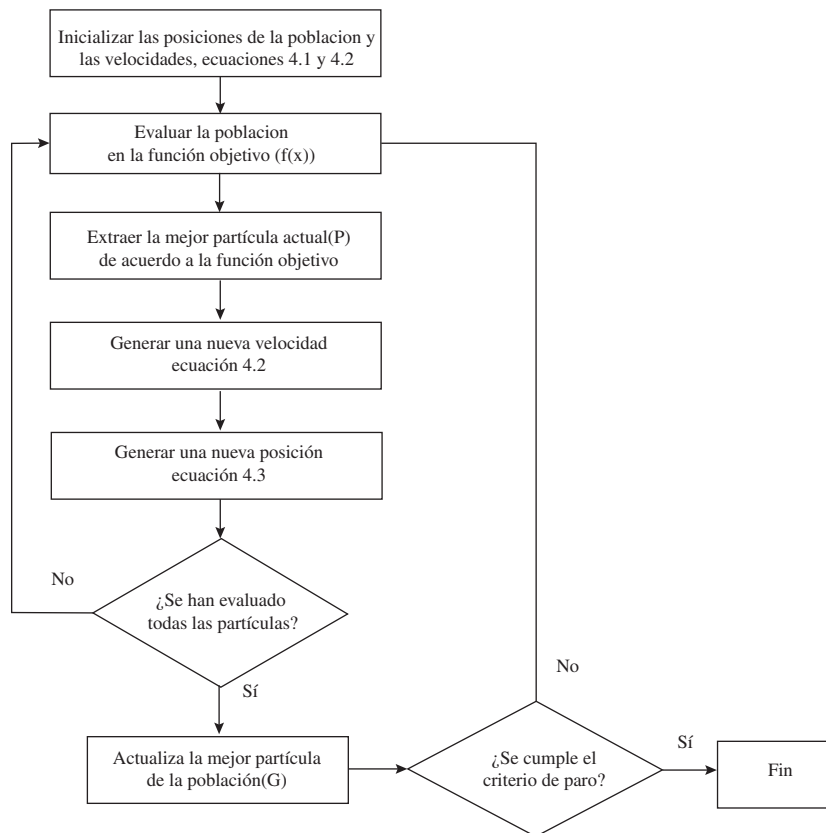


Figura 4.2. Diagrama de flujo del algoritmo PSO.

De acuerdo con la figura 4.2, en el algoritmo PSO primero se genera una población inicial de partículas y con esto también son inicializadas las velocidades de cada una de ellas, mediante las ecuaciones 4.1 y 4.2. Después, las partículas son evaluadas en la función objetivo la cual depende del problema a optimizar. El siguiente paso consiste en extraer la mejor partícula actual, dependiendo de la función objetivo y del problema. Es decir, el máximo o el mínimo. Como siguiente punto, se genera un nuevo vector de velocidades para una partícula en específico, usando la ecuación 4.2. Seguido de esto se genera una nueva posición para dicha partícula, mediante la ecuación 4.3. Estos dos últimos pasos, se repiten para todas las partículas de la población y una vez que toda la población fue evaluada, se verifica el criterio de paro. Si dicho criterio se cumple, el algoritmo termina, si no, se repite el proceso.

4.3 Codificación de PSO

Todas las etapas que se han explicado en las secciones anteriores, se han codificado en MatLAB®. Al igual que las demás implementaciones presentadas en este libro, la de PSO ha sido realizada de forma sencilla, con el propósito de facilitar la comprensión al lector. Para verificar la funcionalidad del algoritmo PSO se usa como función objetivo la ecuación de Rosenbrock^[37], que con frecuencia es considerada para analizar el comportamiento de las técnicas metaheurísticas. La función de Rosenbrock y sus características se describen en la ecuación 4.4.

$$f(x) = \sum_{i=1}^{d-1} \left[100 (x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right], \quad x_i \in [-5, 10] \quad (4.4)$$

Donde: el espacio de búsqueda está acotado por los límites $l = -5$ y $u = 10$, d corresponde al número de dimensiones, para el ejemplo propuesto $d = 2$. La ecuación de Rosenbrock es un problema de minimización, que tiene como el óptimo global en $f(\mathbf{x}) = 0$ con $\mathbf{x} = [1, 1]$ para $d = 2$. La superficie de búsqueda para este problema se muestra en la figura 4.3, mientras que la codificación en MatLAB® se describe en el programa 4.1.

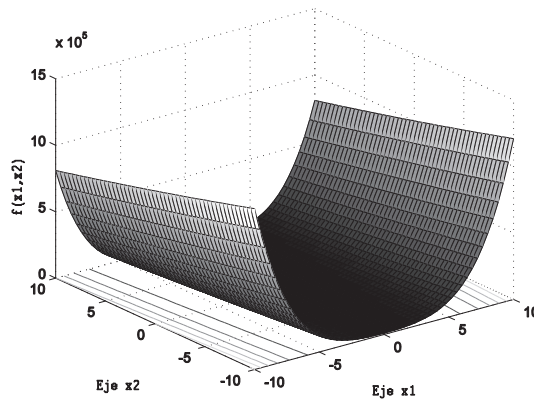


Figura 4.3. Superficie de búsqueda de la función de Rosenbrock.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Función de Rosenbrock
%Erik Cuevas, Valentín Osuna-Enciso, Diego Oliva, Margarita Díaz
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Función que recibe un vector x n-dimensional
function fx = rosenbrock(x)
%Número de dimensiones 2 para este problema
n = 2;
sum = 0;
%Función de Rosenbrock
for j = 1:n-1;
    sum = sum+100*(x(j)^2-x(j+1))^2+(x(j)-1)^2;
end
%Regresa el valor de la función objetivo
fx = sum;

```

Programa 4.1. Código en MatLAB® para la función de Rosenbrock.

El código para PSO se describe en el programa 4.2. Para fines prácticos y con el propósito de evitar complicaciones al lector, se ha agregado una fase adicional, donde se verifica que las nuevas posiciones obtenidas en cada iteración, no salgan de las dimensiones del espacio de búsqueda.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%PSO Estándar, propuesto por Kennedy y Eberhart [31]
%Prueba con función de Rosenbrock
%Erik Cuevas, Valentín Osuna-Enciso, Diego Oliva, Margarita Díaz
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Limpiar memoria y cerrar ventanas de MatLAB
close all
clear all
%dimensiones del espacio de búsqueda
d = 2;
%límites del espacio de búsqueda
%límite inferior
l = [-5, -5];
%límite superior
u = [10, 10];
%Máximo número de iteraciones de PSO
Max_iter = 500;
%Cantidad de partículas en la población
Part_N = 50;

```

```

%Proceso de inicialización de PSO Ecuación 4.1
x = l(1) + rand(Part_N,d).*(u(1) - l(1));
%Evalúa la función objetivo
for i = 1:Part_N
    obj_func(i,:) = rosenbrock(x(i,:));
end
%Obtiene el mejor valor global (mínimo)
[glob_opt, ind] = min(obj_func);
%Vector de óptimo global
%Vector de unos del tamaño de la población
G_opt = ones(Part_N,d);
%Valores del óptimo en dimensión 1
G_opt(:,1) = x(ind,1);
%Valores del óptimo en dimensión 2
G_opt(:,2) = x(ind,2);
Mejor_pos = [x(ind,1),x(ind,2)];
%Mejor local para cada partícula
Loc_opt = x;
%Inicializa velocidades
v = zeros(Part_N,d);
%inicia proceso de optimización PSO
t = 1;
%Criterio de paro
while t < Max_iter
    %Calcula la nueva velocidad ecuación 4.2
    v = v + rand(Part_N,d).*(Loc_opt - x) + rand(Part_N,d).*(G_opt - x);
    %Calcula nueva posición ecuación 4.3
    x = x + v;
    for i = 1:Part_N
        %Se verifica que las partículas no se salgan de los límites u y l
        if x(i,1) > u(1)
            x(i,1) = u(1);
        elseif x(i,1) < l(1)
            x(i,1) = l(1);
        elseif x(i,2) > u(2)
            x(i,2) = u(2);
        elseif x(i,2) < l(2)
            x(i,2) = l(2);
        end
        %Se evalúan las nuevas posiciones en la función objetivo
        Nva_obj_func(i,:) = rosenbrock(x(i,:));
        %Se verifica si se actualizan los óptimos locales
        if Nva_obj_func(i,:) < obj_func(i,:)
            %Actualiza óptimo local
            Loc_opt(i,:) = x(i,:);
            %Actualiza función objetivo
            obj_func(i,:) = Nva_obj_func(i,:);
        end
        %Obtiene el mejor valor global (mínimo)
        [Nvo_glob_opt, ind] = min(obj_func);
    end
    t = t + 1;
end

```



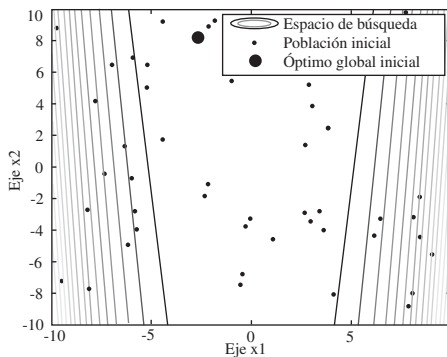
```

%Se verifica si se actualiza el óptimo global
if Nvo_glob_opt < glob_opt
    glob_opt = Nvo_glob_opt;
%Valores del óptimo en dimensión 1
    G_opt(:,1) = x(ind,1);
%Valores del óptimo en dimensión 2
    G_opt(:,2) = x(ind,2);
    Mejor_pos = [x(ind,1),x(ind,2)];
end
%Almacena los valores de función objetivo en cada iteración
    Evol_func_obj(t) = glob_opt;
%Incrementa iteraciones
    t = t + 1;
end
%Grafica la evolución de la función objetivo
plot(Evol_func_obj)
disp(['Mejor posición x: ',num2str(Mejor_pos)])
disp(['Mejor valor función objetivo: ',num2str(glob_opt)])

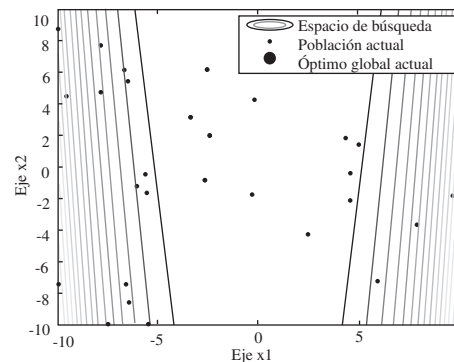
```

Programa 4.2. Código en MatLAB® para el algoritmo PSO.

Un proceso adicional permite que en cada iteración sea almacenado el óptimo global, que se grafica al final del proceso de optimización. En la figura 4.4(a) se presenta una vista superior de la superficie de la función objetivo, donde además se grafican la población y el óptimo global al ser inicializados. En la figura 4.4(b) aparece la población y el óptimo global actuales tras las primeras 125 iteraciones. Mientras que en la figura 4.4(c) se grafican tanto la población final como el óptimo global al término del proceso de optimización. Finalmente la evolución de los mejores valores de la función objetivo, se presenta en la figura 4.4(d), para obtener estos resultados se han empleado 50 partículas y 250 iteraciones. Estos valores pueden calibrar de diversas maneras, en este capítulo han sido elegidos tras diversos experimentos realizados. La mejor posición del óptimo global obtenido es $x = [0.9936, 0.992]$ con un valor de $f(x) = 0.002$, los cuales son muy cercanos a los valores esperados para este problema.



a)



b)

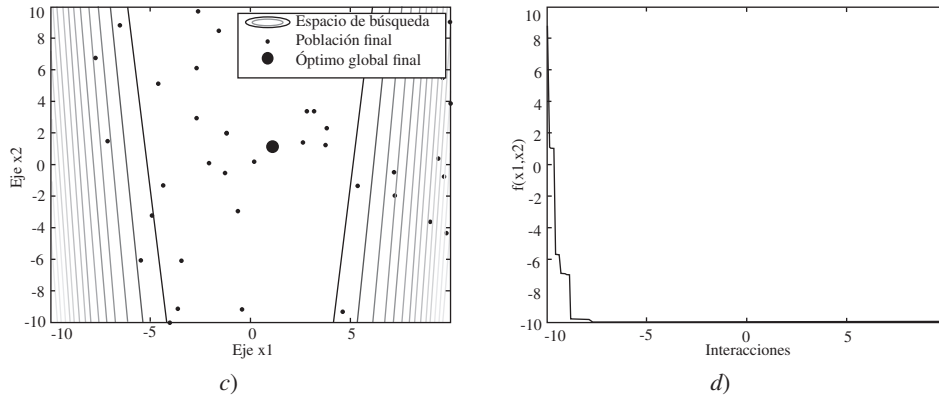


Figura 4.4. Ejemplo de optimización usando el algoritmo PSO: *a)* población y óptimo global después de ser inicializados. *b)* población y óptimo global tras 125 iteraciones. *c)* población y óptimo global al final de las iteraciones. *d)* evolución de la función objetivo.

4.4 Variantes de PSO

Como se ha analizado en las secciones anteriores el algoritmo PSO es una alternativa eficiente para diversos problemas de optimización. Sin embargo, algunas investigaciones han comprobado que este método puede quedar atrapado en óptimos locales^[38], lo cual afecta directamente su desempeño. Con el objetivo de contrarrestar esta deficiencia, se han generado un gran número de variantes del PSO. Algunas de ellas se basan en la modificación de la etapa de inicialización de la población. Otras incluyen nuevos parámetros que permiten generar un balance entre la exploración ya explotación. Mientras que algunos investigadores han propuesto diferentes técnicas que incluyen vectores de inercia u operaciones de mutación usando el óptimo global tras cada iteración. También se han hecho propuestas para que el PSO pueda resolver problemas de optimización con restricciones. Sin embargo, es importante aclarar que no todas las variantes mejoran completamente el desempeño de PSO, para esto se debe buscar cuál se adapta mejor al problema que se pretende resolver. En esta sección se analizarán las variantes más importantes de PSO que se han encontrado tras un análisis minucioso.

4.4.1 Variantes en el proceso de inicialización

Dentro de las diferentes técnicas de inicialización que se proponen una que resulta muy interesante, presenta el uso del “**Opposition-Based Learning, OBL**”^{[39], [40]}, el cual es un método de aprendizaje automático, que se encarga de encontrar los valores opuestos de una población en el espacio de búsqueda. De acuerdo con los autores de OBL un número opuesto se puede definir por medio de la ecuación 4.5.

$$\bar{x}_j = u_j + l_j - x_j \quad (4.5)$$

Para la ecuación 4.5, \bar{x}_j corresponde al valor opuesto del elemento de la población, mientras que x_j es un elemento de la población que previamente ha sido inicializado (por ejemplo: aleato-

riamente). Finalmente u_j y l_j corresponden a los límites superior e inferior del espacio de búsqueda en la dimensión j , respectivamente. Una vez que las posiciones opuestas de las partículas son calculadas, éstas deben evaluarse en la función objetivo. Las mejores partículas existentes entre la población generada de manera aleatoria y sus valores opuestos, son las que serán optimizadas por los operadores de PSO.

$$x = \begin{cases} \bar{x} & \text{si } f(\bar{x}) < f(x) \\ x & \text{si otro caso} \end{cases} \quad (4.6)$$

La ecuación 4.6 presenta la regla empleada para elegir los valores que existen entre ambas poblaciones. Aunque en este caso se presenta para minimizar la función objetivo, sólo cambiando el signo se puede adaptar para un problema de maximización. Es importante mencionar que el OBL se puede incorporar en diversas etapas del proceso de optimización y no solamente en la inicialización. Es decir, cada vez que la población sufre alguna modificación, se pueden sus valores opuestos.

En la figura 4.5 se muestra una descripción gráfica del proceso de optimización usando el algoritmo OBL. En la figura 4.5 (a), se presenta como ejemplo el espacio de búsqueda de una función matemática de prueba en tres dimensiones. Para este problema se pretende encontrar el valor máximo de dicha función usando OBL. Por lo tanto, en la figura 4.5 (b), se tienen tres conjuntos de posibles soluciones. La población P (línea punteada), corresponde al conjunto de soluciones iniciales. Mientras que la población OP (línea con guiones) se refiere al conjunto de soluciones opuestas, que han sido calculadas con la ecuación 4.5. Por lo tanto, al comparar ambos conjuntos usando la ecuación 4.6, se genera un nuevo conjunto P' que contiene las mejores soluciones existentes entre las soluciones iniciales y las opuestas. De tal manera que este nuevo conjunto puede ser modificado por medio de los operadores del algoritmo PSO.

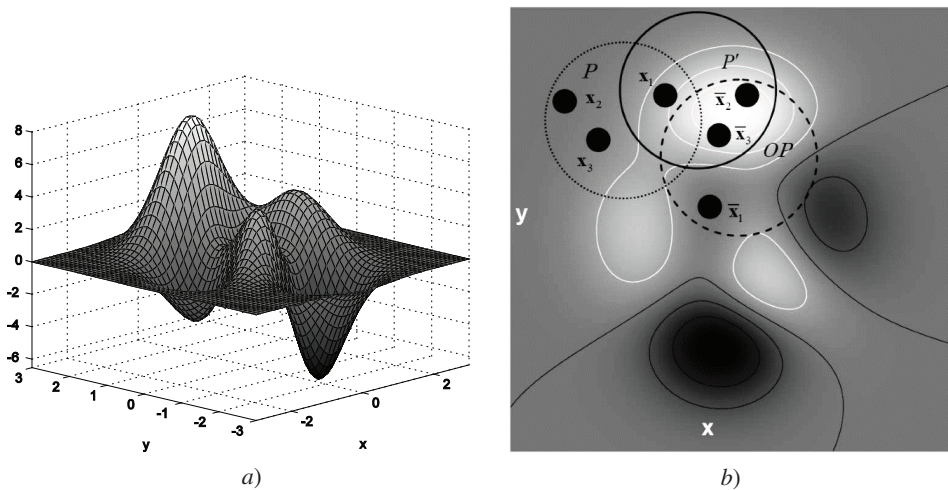


Figura 4.5. Ejemplo gráfico del funcionamiento del OBL.

Además del uso de OBL como estrategia de inicialización, dentro de las variantes de PSO existen una gran cantidad de investigaciones, que se han llevado a cabo empleando diferentes secuencias matemáticas. Ejemplo de esto se presenta en la referencia ^[41], donde los autores hacen un estudio de

los efectos que se tienen al iniciar la población con las secuencias de Halton, Sobol y Faure. Aquí se logra comprobar que la secuencia de Sobol es la que da mejores resultados empleando PSO.

4.4.2 Variantes en la velocidad de las partículas >

Otras modificaciones que comúnmente se llevan a cabo para incrementar el rendimiento del algoritmo PSO, consisten en generar nuevas ecuaciones para el cálculo de la velocidad de las partículas. Una de las técnicas más usadas para calcular la velocidad se presenta en la ecuación 4.7.

$$\mathbf{v}^{t+1} = \mathbf{v}^t + \alpha \varepsilon_1 e[\mathbf{P} - \mathbf{X}^t] + \beta \varepsilon_2 e[\mathbf{G} - \mathbf{X}^t] \quad (4.7)$$

Donde ε_1 y ε_2 son dos vectores cuyos elementos son valores aleatorios que se modifican en cada iteración y se encuentran distribuidos uniformemente entre 0 y 1. Por otra parte, se emplea el producto de Hadamart existente entre dos matrices y se define por el operador e . Los parámetros α y β , son conocidos como las constantes de aceleración o parámetros de aprendizaje, de forma experimental en la literatura es común encontrar que estos valores están establecidos de la siguiente manera:

$$\alpha \approx \beta \approx 2 \quad (4.8)$$

Algoritmo PSO acelerado

Otra modificación que es común encontrar para PSO consiste en incluir una variable adicional a la ecuación 4.7 con el objetivo de “acelerar” el proceso de optimización y la convergencia. Esta modificación se conoce como función de inercia θ , aquí el vector \mathbf{v}^t de la ecuación 4.7 es sustituido por $\theta \mathbf{v}^t$. De tal manera, que la ecuación 4.7 puede ser reescrita como sigue:

$$\mathbf{v}^{t+1} = \theta \mathbf{v}^t + \alpha \varepsilon_1 e[\mathbf{P} - \mathbf{X}^t] + \beta \varepsilon_2 e[\mathbf{G} - \mathbf{X}^t] \quad (4.9)$$

Aquí θ toma valores aleatorios distribuidos entre 0 y 1. Sin embargo, en numerosas aplicaciones en la literatura, se considera este valor como una constante que toma los valores como se menciona en la ecuación 4.10.

$$\theta \approx 0.5 : 0.9 \quad (4.10)$$

Para continuar con la analogía de PSO, el uso de θ es equivalente a tener una masa que estabiliza el movimiento de las partículas, con esto se espera que el algoritmo tenga una más rápida convergencia.

Otra interesante propuesta se refiere al uso de solo el mejor global (\mathbf{G}) en lugar de usar también, las mejores posiciones actuales (\mathbf{P}). De tal manera que se simplifica la forma en que se calcula el vector de velocidad y queda definido por la ecuación 4.11.

$$\mathbf{v}^{t+1} = \mathbf{v}^t + \alpha (\varepsilon - 1/2) + \beta (\mathbf{G} - \mathbf{X}^t) \quad (4.11)$$

Donde nuevamente el valor de ε es aleatorio y uniformemente distribuido entre 0 y 1. Mientras que el factor de 1/2 sólo se usa por conveniencia matemática. En esta misma modificación denominada PSO Acelerado, también se ha propuesto una nueva ecuación para desplazar las partículas

hacia sus nuevas posiciones. De esta manera, la ecuación 4.3 se puede reescribir como a continuación se muestra:

$$\mathbf{x}^{t+1} = (1 - \beta)\mathbf{X}^t + \beta\mathbf{G} + \alpha(\mathbf{v} - 0.5) \quad (4.12)$$

Los valores típicos que se usan para los parámetros de la ecuación 4.12 se encuentran en los rangos: $\alpha \approx 0.1 : 0.4$ y $\beta \approx 0.1 : 0.7$. Sin embargo, para fines prácticos en la literatura es común encontrar que los valores que se usan para ambas variables son $\alpha \approx 0.2$ y $\beta \approx 0.5$, para funciones unimodales. Es importante mencionar que tanto α como β tienen como propósito escalar los valores de las nuevas posiciones en el espacio de búsqueda.

4.4.3 Otras mejoras al algoritmo PSO >

Existe una gran cantidad de modificaciones del algoritmo PSO y hasta ahora se han analizado las más relevantes. Sin embargo hay dos vertientes que son importantes, una de ellas es la estimación de parámetros para reducir la aleatoriedad (PSO acelerado) y la segunda es el uso de diversas reglas para poder resolver problemas de optimización con restricciones.

Mejora del PSO acelerado

Es bien sabido que la mayoría de los algoritmos de optimización metaheurísticos tienen parámetros que deben ser ajustados para que se tenga un buen desempeño. La versión estándar de PSO cuenta con pocos parámetros que deben ser calibrados. Sin embargo, la mayoría de las modificaciones involucran el uso de nuevos parámetros, tal es el caso del PSO acelerado que se presenta en la sección 4.2.1. Existen en la literatura referente, diversas propuestas para la estimación automática de estos parámetros. Ejemplo de esto es la técnica que se presenta para calcular el valor de α . Además de esto, con dicha modificación se reduce la aleatoriedad que pueda existir en los cálculos. Para esto se emplea una función monótona decreciente de la siguiente forma:

$$\alpha = \alpha_0 e^{-\gamma} \quad (4.13)$$

Esta ecuación 4.13 también puede definirse como:

$$\alpha = \alpha_0 - \gamma', \quad (0 < \gamma < 1) \quad (4.14)$$

Para ambas ecuaciones el valor inicial del parámetro de aleatoriedad se encuentra en el rango: $\alpha_0 \approx 0.5 : 1$. Mientras que t es el número de iteraciones del algoritmo y $(0 < \gamma < 1)$ es un parámetro de control. Un ejemplo de esto consiste en elegir un valor de $\alpha_0 = 0.7$ y $\gamma = 1$ de y $t = 10$ este modo:

$$\alpha = 0.7^{10} \quad (4.15)$$

4.5 PSO para optimización con restricciones

Las técnicas explicadas hasta ahora han sido para problemas de optimización sin restricciones. Sin embargo, en la práctica la mayoría de los problemas tienen restricciones. Por tal motivo un algoritmo metaheurístico robusto debe ser capaz de encontrar las soluciones de este tipo de problemas.

El algoritmo PSO en un principio fue creado para trabajar con problemas sin restricciones. En la literatura referente existe una gran cantidad de modificaciones propuestas para que PSO pueda ser un optimizador capaz de encontrar las soluciones a dichos problemas.

De forma general, un problema de optimización (minimización) con restricciones se define de la siguiente manera:

$$\begin{aligned} \underset{x \in I_n}{\text{Minimizar}} \quad & f_i(\mathbf{x}), \quad i = 1, 2, \dots, M, \\ \text{sujeto a} \quad & \phi_j(\mathbf{x}) = 0, \quad j = 1, 2, \dots, J, \\ & \psi_k(\mathbf{x}) \leq 0, \quad k = 1, 2, \dots, K, \end{aligned} \quad (4.16)$$

Donde $f_i(\mathbf{x})$ corresponde al conjunto de funciones objetivo, $\phi_j(\mathbf{x})$ y $\psi_k(\mathbf{x})$ son las restricciones del problema. De forma gráfica, la figura 4.6 presenta la superficie de búsqueda de un problema de optimización con restricciones. Se tiene un espacio acotado por los límites superior e inferior y de entro de él se define la función objetivo $f_i(\mathbf{x})$. También dentro de este espacio se definen las restricciones, de tal manera que las soluciones del problema se encuentran en una sección reducida del espacio de búsqueda.

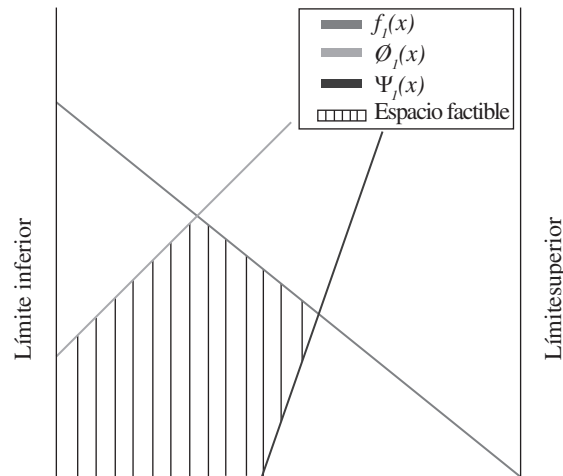


Figura 4.6. Optimización con restricciones.

Como ya se mencionó, no hay una única forma de encontrar el valor óptimo de este tipo de problemas. Pero, sí existe un conjunto de técnicas que son ampliamente usadas y con las que se ha modificado el algoritmo PSO. Sin embargo, por su simplicidad una de las más usadas es el de descartar soluciones. El proceso consiste en verificar que cada partícula satisfaga todas las restricciones para poder ser evaluada en la función objetivo. Si alguno de los elementos de la población no satisface las restricciones, se genera una nueva partícula y este proceso se lleva a cabo hasta que se encuentre una posible solución que cumpla con las restricciones. Para generar cada nueva solución, se puede hacer mediante los operadores propios de PSO o de manera aleatoria, pero siempre debe verificarse que las restricciones se cumplan antes de evaluar la solución en la función objetivo. En la figura 4.7 se presenta el diagrama de flujo de esta técnica.

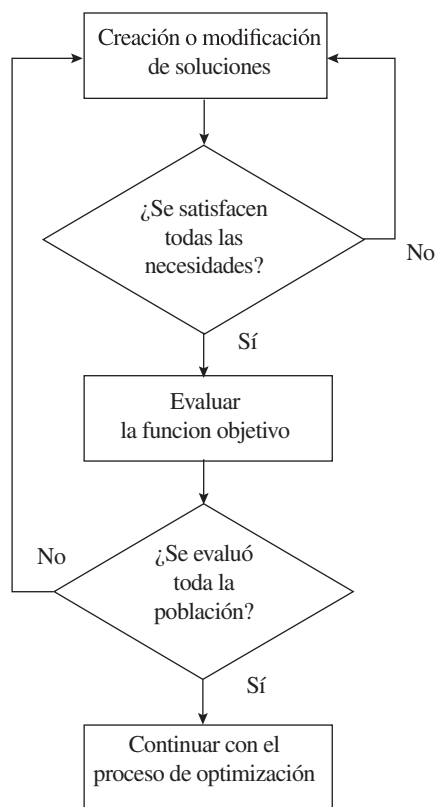


Figura 4.7. Diagrama de flujo para la técnica de descartar soluciones en optimización con restricciones.






Otro popular método para afrontar los problemas de la optimización con restricciones, es convertirlo en un problema sin restricciones. Para eso se emplea el método de penalización en el que se hace uso de los multiplicadores de Lagrange. A continuación se muestra un ejemplo de este tipo de funciones de penalización:

$$F(\mathbf{x}) = f(\mathbf{x}) + \lambda g(\mathbf{x})^2 \quad (4.17)$$

Donde $F(\mathbf{x})$ es la nueva función objetivo sin restricciones, que contiene a la función objetivo original $f(\mathbf{x})$ con una restricción definida por $g(\mathbf{x})$. El parámetro de penalización se define como $\lambda \geq 1$. De este modo el problema se debe resolver como cualquier otro de los problemas de optimización tratados en este libro. Es importante mencionar que para cada una de las restricciones se debe definir un valor de penalización distinto.

Por último, una regla muy simple empleada para este tipo de problemas consiste en asignar un valor de función objetivo considerado como “pésimo” al miembro de la población que no cumpla con las restricciones del problema. De tal manera que para un problema de optimización, la partícula que no satisfaga alguna restricción tendrá un valor grande. Con esto lo que se pretende es sólo considerar las soluciones que se encuentran dentro del área de búsqueda factible.

Ejercicios

4.1 	<p>Considerando el proceso de optimización de PSO, proponga una mejora a la velocidad de las partículas teniendo en cuenta el concepto de inercia. Para esto, reescriba el código de PSO \mathbf{V}^t por $\theta \mathbf{V}^t$ en la ecuación 4.2, para esto considere el valor de $\theta \in [0, 1]$. Observe si mejora la convergencia o el tiempo computacional.</p>
4.2 	<p>Escriba o modifique el algoritmo PSO presentado en este capítulo, para que optimice la siguiente función objetivo multidimensional:</p> <p>Minimizar $f(\mathbf{x}) = -\sum_{i=1}^4 \alpha_i \exp\left(-\sum_{j=1}^6 A_{i,j} (x_j - P_{ij})^2\right)$, donde $\alpha = [1.0, 1.2, 3.0, 3.2]^T$</p> $\mathbf{A} = \begin{bmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix}, \quad \mathbf{P} = 10^{-4} \begin{bmatrix} 1312 & 1696 & 5596 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{bmatrix}$ <p>Considerando los límites para cada una de las 6 dimensiones son: $0 \leq x \leq 1$</p>
4.3 	<p>Considerando la modificación realizada en el ejercicio 4.1 y la función objetivo propuesta en el ejercicio 4.2, compare el desempeño de ambos algoritmos con base en la evolución de la función objetivo.</p>
4.4 	<p>En base a la ecuación 4.2, ¿Cómo modificaría el algoritmo PSO para evitar que quede atrapado en óptimos locales?</p>
4.5 	<p>Escribe o modifica un programa en Matlab, en el cual el algoritmo PSO encuentre la solución global de la siguiente función objetivo:</p> <p>Minimizar $f(x) = \sum_{j=1}^n \left[\sum_{i=1}^n (i^j + \beta) \left[\left(\frac{x_i}{i} \right)^j - 1 \right] \right], \quad -n \leq x_i$</p> <p>Donde $\beta > 0$</p> <p>¿El valor de β afecta el valor de la solución óptima?</p>