3

TÉCNICAS METAHEURÍSTICAS. ALGORITMOS BASADOS EN NUBES DE PARTÍCULAS

3.1 INTRODUCCIÓN.

En la vida cotidiana, continuamente se presentan y resuelven problemas de optimización. Los pequeños problemas se resuelven utilizando la lógica y pequeñas fórmulas matemáticas. Pero si los problemas presentan complejidad o son de una elevada magnitud se ha de recurrir al uso de computadores.

Uno de los principales objetivos de cualquier Ingeniería es el uso de métodos exactos y heurísticos para optimizar funciones objetivo. La optimización de estos problemas parte de un conjunto de datos y una serie de condiciones y limitaciones que dificultan la utilización de métodos exactos. La dificultad se presenta principalmente por la alta complejidad de los cálculos y la duración de éstos, en algunas ocasiones el tiempo de resolución está limitado.

Son diversos los problemas de optimización complejos y las limitaciones en su resolución, esto ha provocado el desarrollo de técnicas metaheurísticas [63]. Una metaheurística es un conjunto de conceptos que se usan para definir métodos heurísticos que pueden ser aplicados a una amplia variedad de problemas, es decir, una metaheurística es vista como un marco general algorítmico que se puede aplicar a diferentes problemas de optimización con mínimos cambios para ser adaptado a un problema específico.

Los algoritmos metaheurísticos son procedimientos iterativos que guían una heurística subordinada, combinando de forma inteligente distintos conceptos para explorar y explotar adecuadamente el espacio de búsqueda. Son algoritmos aproximados de optimización y búsqueda de propósito general.

Las técnicas metaheurísticas son capaces de proporcionar muy buenas soluciones (no necesariamente la óptima pero si aproximada) en tiempo y con recursos razonables.

Una de las técnicas metaheurísticas son los algoritmos basados en nubes de partículas [95, 96, 97], en inglés Particle Swarm Optimization (PSO en adelante). Estos algoritmos son muy recientes y novedosos, y pertenecen al grupo de los algoritmos bioinspirados (metaheurísticas que se inspiran en la naturaleza para resolver problemas) [154].

Los padres del PSO son James Kennedy y Russ C. Eberhart. En 1995, desarrollaron el algoritmo originario experimentando con algoritmos que modelaban el comportamiento del vuelo de algunos pájaros o en el movimiento de los bancos de peces. El algoritmo se basa en la siguiente metáfora social: los individuos que son parte de una sociedad tienen una opinión influenciada por la creencia global compartida por todos los posibles individuos. Cada individuo puede modificar su opinión (o estado) según tres factores: el conocimiento del entorno, los estados por los que ha pasado y los estados por los que han pasado los individuos cercanos.

En el algoritmo PSO, cada individuo, llamado partícula, se va moviendo en un espacio multidimensional que representa su espacio social. Debido a su planteamiento, este tipo de algoritmo se adapta muy bien a problemas matemáticos tanto de carácter continuo como de naturaleza discreta [135].

3.2 OPTIMIZACIÓN CON TÉCNICAS METAHEURÍSTICAS

3.2.1 Técnicas Metaheurísticas

Para la optimización de problemas y cálculos de alta complejidad se han desarrollado múltiples técnicas y métodos. Estas técnicas de optimización se pueden clasificar en técnicas exactas y en técnicas aproximadas.

Las técnicas exactas (enumerativas, exhaustivas, etc.) garantizan encontrar la solución óptima de cualquier problema. Serían los métodos idóneos si no tuvieran el inconveniente de la cantidad de tiempo necesario para la resolución. El tiempo crece exponencialmente con el tamaño del problema. En determinados casos, el tiempo de resolución podría llegar a ser de varios días, meses o incluso años, lo que provoca que el problema sea inabordable con estos métodos.

Las técnicas aproximadas sacrifican la garantía de encontrar el resultado óptimo a cambio de obtener una buena solución en un tiempo razonable. Se han venido desarrollando durante los últimos 30 años y se distinguen tres tipos: métodos constructivos, métodos de búsqueda local y las técnicas metaheurísticas.

Los métodos constructivos suelen ser los más rápidos. Partiendo de una solución vacía, a la que se les va añadiendo componentes, generan una solución completa. Las soluciones ofrecidas suelen ser de muy baja calidad. Su planteamiento depende en gran parte del tipo de problema. Es muy difícil encontrar métodos de esta clase que produzcan buenas soluciones, y en algunas ocasiones es casi imposible, por ejemplo, en problemas con muchas restricciones.

Los métodos de búsqueda local usan el concepto de vecindario y se inician con una solución completa recorriendo parte del espacio de búsqueda hasta encontrar un óptimo local. El vecindario de una solución es el conjunto de soluciones que se pueden construir a partir de aquella aplicando un operador de modificación denominado movimiento. Estos métodos parten de una solución inicial, examinan su vecindario y eligen el mejor vecino continuando el proceso hasta que encuentran un óptimo local. En función del operador de movimiento utilizado, el vecindario cambia y el modo de explorar el espacio de búsqueda también, pudiendo la búsqueda complicarse o simplificarse.

Las técnicas metaheurísticas son algoritmos no exactos. Se fundamentan en la combinación de diferentes métodos heurísticos a un nivel más alto para conseguir una

exploración del espacio de búsqueda más eficaz y eficiente.

Una heurística es una técnica que busca soluciones buenas (óptimas o casi óptimas) a un coste computacional razonable, aunque sin garantizar la factibilidad de las mismas. En algunos casos ni siquiera puede determinar la cercanía al óptimo de una solución factible [143].

Glover y col. introducen por primera vez el término de metaheurística en [62], definen las metaheurísticas como métodos que integran de diversas maneras, procedimientos de mejora local y estrategias de alto nivel para crear un proceso capaz de escapar de óptimos locales y realizar una búsqueda robusta en el espacio de búsqueda. En su evolución, estos métodos han incorporado diferentes estrategias para evitar la convergencia a óptimos locales, especialmente en espacios de búsqueda complejos.

Este tipo de técnicas se caracterizan por las siguientes propiedades:

- Las metaheurísticas son estrategias generales que guían el proceso de búsqueda.
- El objetivo es una búsqueda eficiente que encuentre soluciones casi óptimas.
- Pueden incorporar mecanismos para evitar la exploración en regiones del espacio de búsqueda no óptimas.
- El procedimiento de cualquier metaheurística es genérico, no depende del problema.
- Las metaheurísticas utilizan métodos heurísticos específicos que son controlados por una estrategia de más alto nivel.

- Las metaheurísticas utilizan funciones de bondad para cuantificar el grado de adecuación de una determinada solución.

En resumen, una metaheurística es una estrategia genérica de alto nivel que usa diferentes métodos heurísticos para explorar en busca de una solución óptima o casi óptima en espacios de búsqueda de gran tamaño. La metaheurística debe identificar rápidamente las regiones prometedoras del espacio de búsqueda global, y no malgastar tiempo en regiones que hayan sido exploradas y/o no contienen soluciones de alta calidad.

3.2.2 Clasificación de las técnicas metaheurísticas

Hay diferentes formas de clasificar las técnicas metaheurísticas [30]: basadas en la naturaleza (algoritmos bioinspirados) o no basadas en la naturaleza, basadas en memoria o sin memoria, con función objetivo estática o dinámica, etc.

La clasificación más empleada es la que se basa en si la técnica utiliza un único punto del espacio de búsqueda o trabaja sobre un conjunto o población. Según esta clasificación las metaheurísticas se dividen en las basadas en trayectoria y las basadas en población.

3.2.2.1 Metaheurísticas basadas en trayectoria

Estas técnicas parten de un punto inicial y van actualizando la solución presente mediante la exploración del vecindario, formando una trayectoria. La búsqueda finaliza cuando se alcanza un número máximo de iteraciones, se encuentra una solución con una calidad aceptable, o se detecta un estancamiento del proceso.

A continuación se describen algunas de las técnicas metaheurísticas basadas en trayectoria:

El Enfriamiento Simulado (ES), es una de las metaheurísticas más antigua [99]. ES simula el proceso de recocido de los metales y del cristal. En cada iteración se elige una solución S_1, a partir de la solución actual S_0. Si S_1 es mejor que S_0, S_1 sustituye a S_0 como solución actual. Si S_1 es peor que S_0, se sigue aceptando pero asignándole una determinada probabilidad. El algoritmo permite elegir soluciones peores a la actual para evitar caer en un óptimo local.

La Búsqueda Tabú (BT), es una de las metaheurísticas más utilizadas en problemas de optimización [62]. La BT se basa fundamentalmente en la utilización de un historial de búsqueda, que permite ejecutar su estrategia de análisis y exploración de diferentes regiones del espacio de búsqueda. Este historial o memoria se implementa como una lista tabú. En cada iteración se elige la mejor solución entre las permitidas y se añade a la lista tabú, donde se mantienen las soluciones recientes que se excluyen de las siguientes iteraciones.

La Búsqueda en Vecindario Variable (BVV) [120]. Este algoritmo es muy genérico, con muchos grados de libertad y permite variaciones y modificaciones particulares. Utiliza una estrategia de cambio entre diferentes estructuras del vecindario. Estas estructuras se definen en el comienzo del proceso algorítmico.

La Búsqueda Local Iterada (BLI) [155], se basa en que en cada iteración, a la solución actual se le aplica un cambio o modificación que da lugar a una solución intermedia. A esta nueva solución se le aplica una heurística base para mejorarla que suele ser un método de búsqueda local. Este nuevo óptimo local obtenido por el método de mejora puede ser aceptado como nueva solución actual si pasa un test de aceptación.

3.2.2.2 Metaheurísticas basadas en población

Las técnicas metaheurísticas basadas en población trabajan con un conjunto de de individuos que representan otras tantas soluciones. Su eficiencia y resultado depende fundamentalmente de la forma con la que se manipula la población en cada iteración.

Seguidamente se describen algunas de las técnicas metaheurísticas basadas en población:

Los Algoritmos Evolutivos (AE) [6]. Este grupo de técnicas se inspiran en la capacidad de la evolución de seres o individuos para adaptarlos a los cambios de su entorno. Cada individuo representa una posible solución. El funcionamiento básico de estos algoritmos es el siguiente: La población se genera de forma aleatoria. Cada individuo de la población tiene asignado un valor de su bondad con respecto al problema considerado, por medio de una función de *aptitud*, capacidad, adaptabilidad o estado, también denominada con bastante frecuencia por la palabra inglesa "fitness". El valor de la *aptitud* de un individuo es la información que el algoritmo utilizar para realizar la búsqueda. La modificación de la población se efectúa mediante la aplicación de tres operadores: selección, recombinación y mutación. En estos algoritmos se pueden distinguir la fase de selección, explotación de buenas soluciones, y la fase de reproducción, búsqueda de nuevas regiones. Se debe de mantener un equilibro entre estas dos fases. La política de reemplazo permite la aceptación de nuevas soluciones que no necesariamente mejoran las existentes.

Los algoritmos evolutivos se pueden clasificar en las siguientes tres categorías: Programación Evolutiva (PE) [55], Estrategias Evolutivas (EE) [142], y los Algoritmos Genéticos (AG), que constituyen una de las técnicas más conocidas, y que fueron introducidos por Holland [81]. El funcionamiento de los algoritmos genéticos se describe en el apéndice II.

La Búsqueda Dispersa (BD) [61, 63], se basa en mantener un conjunto relativamente pequeño de soluciones, conjunto de referencia, que contiene buenas soluciones y otras soluciones diversas. A los diferentes subconjuntos de soluciones que se forman se les aplica operaciones de recombinación y mejora.

Los sistemas basados en Colonias de Hormigas, (ACO) [40], se inspiran en el comportamiento de las hormigas cuando buscan comida: inicialmente, las hormigas exploran el área cercana al hormiguero de forma aleatoria. Cuando una hormiga encuentra comida, la lleva al hormiguero. En el camino, la hormiga va depositando una sustancia química denominada feromona que guía al resto de hormigas a encontrar la comida. El rastro de feromona sirve a las hormigas para encontrar el camino más corto entre el hormiguero y la comida. Este rastro es simulado mediante un modelo probabilístico.

Los Algoritmos Basados en Nubes de Partículas o Particle Swarm Optimization (PSO) [95, 96, 97, 98] son técnicas metaheurísticas inspiradas en el comportamiento del vuelo de las bandadas de aves o el movimiento de los bancos de peces. La toma de decisión por parte de cada individuo o partícula se realiza teniendo en cuenta una componente social y una componente individual, mediante las que se determina el movimiento de esta partícula para alcanzar una nueva posición. A continuación se describe con más detalle el algoritmo PSO.

3.3 ALGORITMO PSO

3.3.1 Introducción al algoritmo PSO

Los algoritmos basados en nubes (también enjambre o cúmulos) de partículas se aplican en diferentes campos de investigación para la optimización de problemas complejos. Como ya se ha dicho anteriormente, el algoritmo PSO es una técnica metaheurística poblacional basada en la naturaleza (algoritmo bioinspirado), en concreto, en el comportamiento social del vuelo de las bandadas de aves y el movimiento de los bancos de peces. Es una técnica relativamente reciente [95, 96, 97]. PSO fue originalmente desarrollado en Estados Unidos por el sociólogo James Kennedy y por el ingeniero Russ C. Eberhart en 1995.

Estos autores describen el algoritmo PSO de la siguiente manera [98]: los individuos (partículas) que conviven en una sociedad tienen una "opinión" que es parte del espacio de búsqueda, compartido por todos los individuos. Cada individuo puede modificar su opinión según tres factores:

- El conocimiento del entorno o *ADAPTACIÓN*.
- Experiencias anteriores del individuo o *MEMORIA DEL INDIVIDUO*.
- Experiencias anteriores de los individuos del vecindario o *MEMORIA DEL VECINDARIO*.

Los individuos adaptan o modifican sus opiniones a las de los individuos con más éxito de su entorno. Con el tiempo, los individuos de un entorno tienen un conjunto de opiniones bastante relacionado.

El funcionamiento básico del PSO simula el comportamiento del vuelo de las bandadas de aves en busca de comida. La estrategia lógica a utilizar es seguir al ave que está más cerca de la comida. Cada ave se modela como una partícula con una solución en el espacio de búsqueda que está siempre en continuo movimiento y nunca se elimina o muere.

El PSO es un sistema multiagente. Las partículas son agentes simples que se mueven por el espacio de búsqueda, guardan y posiblemente comunican la mejor solución que han encontrado. El movimiento de las partículas por el espacio está guiado por las partículas que tienen la mejor solución del momento.

Figura 3.1: Ejemplo de "nube". Banco de peces.

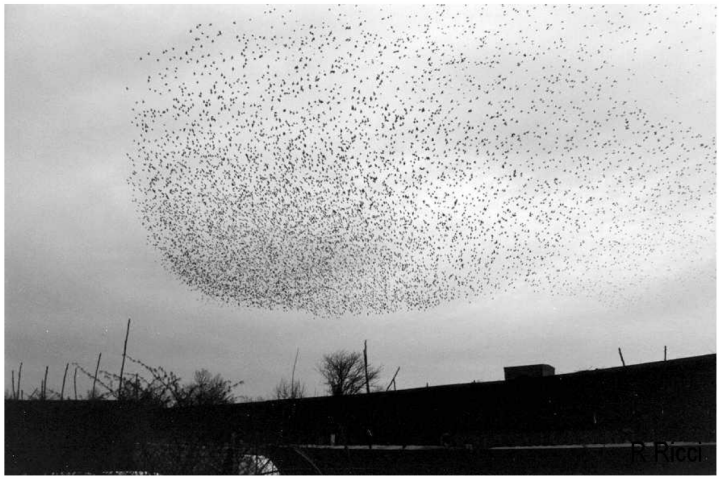

Figura 3.2: Ejemplo de "nube". Bandada de aves.

Las principales características del algoritmo PSO son las siguientes [95]:

- En PSO los agentes de búsqueda (partículas) intercambian información. Las partículas modifican su dirección en función de las direcciones de las partículas de su vecindario.

- PSO almacena la experiencia propia o historia de cada agente. La partícula decide su nueva dirección en función de la mejor posición por la que pasó anteriormente.

- Suele tener una convergencia rápida a buenas soluciones.

- La población del algoritmo se inicia de forma aleatoria y evoluciona iteración tras iteración.

- La búsqueda persigue siempre la solución más óptima posible.

- La búsqueda se basa exclusivamente en los valores de la función objetivo.

- PSO trabaja con la información del problema codificada.

- Es una técnica estocástica referida en fases (inicialización y transformación).

- PSO tiene operadores de movimiento pero no de evolución como la mutación o el cruzamiento.

- PSO no crea nuevas partículas durante su ejecución, sino que siempre son las mismas partículas iniciales modificadas a lo largo del proceso.

3.3.2 Descripción del algoritmo PSO

Se inicia la descripción del algoritmo PSO estudiando la anatomía de la partícula. Una partícula está compuesta de tres vectores y dos valores de *aptitud* (también conocida como bondad, adaptación, capacidad, adecuación o "fitness") con respecto al problema considerado:

1) Tres vectores:

- El vector $\mathbf{x_i} = \left(x_{i,1}, x_{i,2}, ..., x_{i,N}\right)$ almacena la posición actual de la partícula.
- El vector $\mathbf{mejorpos_i} = \left(mejorpos_{i,1}, mejorpos_{i,2}, ..., mejorpos_{i,N}\right)$ almacena la posición de la mejor solución encontrada por la partícula hasta el momento.
- El vector de velocidad $\mathbf{v_i} = \left(v_{i,1}, v_{i,2}, ..., v_{i,N}\right)$ almacena la dirección según la cual se moverá la partícula.

2) Dos valores de *aptitud*:

- El valor de $aptitud\_x_i$, almacena el valor de adaptación o adecuación de la posición actual correspondiente al vector $\mathbf{x_i}$.

- El valor de $aptitud\_mejorpos_i$, almacena el valor de adecuación de la partícula con mejor solución local encontrada hasta el momento, correspondiente al vector $\mathbf{mejorpos_i}$.

La descripción del proceso algorítmico es la siguiente:

1) La nube se inicializa generando las posiciones (de forma aleatoria, regular o combinación de ambas).

2) Se generan las velocidades aleatoriamente en un intervalo establecido $[-v_{max}, v_{max}]$, no es conveniente fijarlas a cero [98].

3) Se calcula la *aptitud* de cada partícula y se actualizan los valores de $aptitud\_x_i$ y $aptitud\_mejorpos_i$.

4) Las partículas se mueven en cada iteración desde una posición del espacio de búsqueda hasta otra. Al vector de posición $\mathbf{x_i}$ se le añade el vector velocidad $\mathbf{v_i}$ para obtener un nuevo vector $\mathbf{x_i}$.

5) Con la nueva posición de la partícula se calcula y actualiza *aptitud_*x_i.

6) Si el nuevo valor de *aptitud* es el mejor encontrado por la partícula *i* hasta el momento, se actualizan los valores de *mejorpos*$_i$ y *aptitud_mejorpos*$_i$.

7) Si el nuevo valor de *aptitud_mejorpos*$_i$ es el mejor encontrado por la nube de partículas hasta el momento, se actualizan el valor de la mejor posición de la nube *mejorpos* y su *aptitud_mejorpos*.

8) El vector velocidad de cada partícula es modificado en cada iteración utilizando la velocidad anterior, un componente cognitivo y un componente social. El modelo matemático resultante, y que representa el corazón del algoritmo PSO, viene representado por las siguientes ecuaciones:

$$\mathbf{v}_i^t \leftarrow \omega^{t-1} \cdot \mathbf{v}_i^{t-1} + \varphi_1 \cdot rand_1 \cdot (\mathbf{mejorpos}_i - \mathbf{x}_i^{t-1}) + \varphi_2 \cdot rand_2 \cdot (\mathbf{mejorpos} - \mathbf{x}_i^{t-1}) \quad (3.1)$$

$$\mathbf{x}_i^t \leftarrow \mathbf{x}_i^{t-1} + \mathbf{v}_i^t \quad (3.2)$$

Para $i = 1, 2, \ldots, P$

Donde:

- $\mathbf{x}_i^t$ ≡ vector posición de la partícula *i* en la iteración *t*.
- $\mathbf{v}_i^t$ ≡ vector velocidad de la partícula *i* en la iteración *t*.
- ω^t ≡ factor de inercia en la iteración *t*.
- φ_1, φ_2 ≡ son pesos que controlan los componentes cognitivo y social.
- $rand_1$ ≡ número aleatorio entre 0 y 1.
- $rand_2$ ≡ número aleatorio entre 0 y 1.

- ***mejorpos***$_i$ ≡ mejor posición encontrada por la partícula *i* hasta el momento que posee la mejor solución.
- ***mejorpos*** ≡ representa la posición de la partícula con la mejor solución o *aptitud*.
- *P* ≡ número de partículas que componen la nube.

La ecuación (3.1) actualiza el vector velocidad de cada partícula *i* en la iteración *t*. Igualmente, la ecuación (3.2) actualiza el vector de posición de la partícula *i* para cada iteración.

El primer término de la ecuación (3.1) es el vector velocidad de la anterior iteración, lo que indica que el algoritmo PSO tiene memoria.

El componente cognitivo indica la decisión que tomará la partícula y depende de su propia experiencia, dicho de otra manera, representa la distancia entre la posición actual y la mejor conocida por esa partícula. El componente cognitivo en la ecuación (3.1) es el factor: $\varphi_1 \cdot rand_1 \cdot (\mathbf{mejorpos}_i - \mathbf{x}_i^{t-1})$.

El componente social apunta la decisión que tomará la partícula en base a la influencia del resto de partículas que componen la nube, es decir, representa la distancia entre la posición actual y la mejor posición encontrada por vecindario. El componente social se modela en la ecuación (3.1) como: $\varphi_2 \cdot rand_2 \cdot (\mathbf{mejorpos} - \mathbf{x}_i^{t-1})$.

En la figura 3.3 se muestra el movimiento de una partícula en el espacio de soluciones. Las flechas de línea verde discontinua representan la dirección de los componentes cognitivo y social. La flecha azul discontinua representa la velocidad actual de la partícula. La flecha de línea continua representa la dirección que toma la partícula para moverse desde la posición actual $\mathbf{x}_i^{t-1}$ hasta la nueva posición $\mathbf{x}_i^t$.

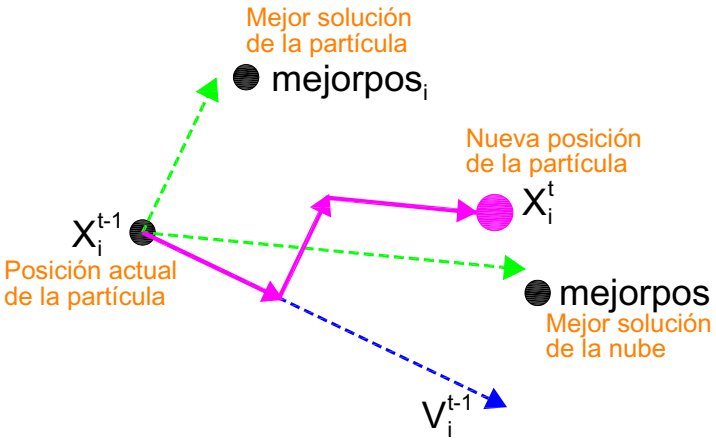

Figura 3.3: Representación gráfica del movimiento de una partícula.

Para conseguir un buen funcionamiento y eficiencia del algoritmo PSO, el valor de la velocidad no debe llegar a ser muy grande durante la ejecución. Limitando la velocidad máxima de cada vector velocidad ($v_{máx}$) y reduciendo gradualmente su valor se consigue mejorar el rendimiento del algoritmo. El control del vector velocidad se puede realizar mediante el ajuste dinámico del factor de inercia [42]. El factor inercia, ω, se puede ir reduciendo progresivamente en cada iteración aplicando la siguiente ecuación:

$$\omega^t = \omega_{\max} - \frac{\omega_{\max} - \omega_{\min}}{t_{\max}} \cdot t \tag{3.3}$$

Donde:

- $\omega_{\max}$ ≡ Factor de inercia inicial.
- $\omega_{\min}$ ≡ Factor de inercia final.
- $t_{\max}$ ≡ Número de iteraciones máximo.
- t ≡ Número de iteración actual.

El tamaño de la nube de partículas juega también un papel importante, ya que, determina el equilibro entre la calidad de las soluciones obtenidas y el número de iteraciones necesarias hasta llegar a una buenas solución (tiempo computacional).

Por último, la interacción de las partículas con el resto depende de su entorno. Se pueden distinguir dos tipos de entornos:

- Geográficos: Calculan las distancias entre la partícula considerada y el resto. El entorno de cada partícula lo componen las más cercanas.

- Sociales: Definen para cada partícula una listado de partículas vecinas, la elección es independientemente de su posición en el espacio. Son los más empleados.

Cuando el tamaño del entorno es toda la nube de partículas, el entorno es a la vez geográfico y social.

3.4 TIPOS DE ALGORITMOS PSO

Se pueden obtener diferentes tipos de PSO atendiendo a diversos factores de configuración:

A) Según la importancia de los pesos cognitivo y social:

- Completo: $\varphi_1, \varphi_2 > 0$. Tanto el componente cognitivo como el social intervienen en el movimiento.

- Cognitivo: $\varphi_1 > 0$ y $\varphi_2 = 0$. Sólo interviene el componente cognitivo en el movimiento.

- Social: $\varphi_1 = 0$ y $\varphi_2 > 0$. Sólo interviene el componente social en el movimiento.

- Social exclusivo: $\varphi_1 = 0$, $\varphi_2 > 0$ y $\mathbf{mejorpos} \neq \mathbf{x}_i$. La posición de la partícula en sí no puede ser la mejor de su entorno.

Para el PSO completo se recomiendan valores de $\varphi_1 = 2$ y $\varphi_2 = 2$, con ellos se obtiene una mayor eficacia y eficiencia del algoritmo en base a las experiencias y ensayos realizados [98].

B) Según el tipo de vecindario utilizado, es decir, la cantidad y posición de las partículas que intervienen en el cálculo de la distancia en la componente social:

- PSO Local, se calcula la distancia entre la posición actual de la partícula y la posición de la mejor partícula perteneciente al entorno local de aquella. El entorno local consiste en las partículas inmediatamente cercanas.

- PSO Global, la distancia se obtiene entre la posición actual de la partícula y la posición de la mejor partícula considerando la nube completa.

3.5 PSEUDOCODIGOS DEL ALGORITMO PSO

3.5.1 Pseudocódigo del algoritmo PSO clásico

El algoritmo PSO trabaja de forma iterativa modificando una nube de partículas mediante la aplicación de movimiento a cada una de ellas. El siguiente pseudocódigo describe el algoritmo PSO clásico:

ALGORITMO 1 – PSO CLASICO

$t = 0$

Nube ← Inicializar Nube de Partículas

Mientras no se alcance la condición de parada **hacer**

$t = t + 1$

Para i = 1 hasta tamaño (*Nube*) **hacer**

Evaluar cada partícula $\boldsymbol{x}_i$ de la *Nube*

Si $aptitud\_x_i$ es mejor que $aptitud\_mejorpos_i$ **entonces**

$mejorpos_i \leftarrow x_i$;

$aptitud\_mejorpos_i \leftarrow aptitud\_x_i$

Fin Si

Si $aptitud\_mejorpos_i$ es mejor que $aptitud\_mejorpos$ **entonces**

$mejorpos \leftarrow mejorpos_i$;

$aptitud\_mejorpos \leftarrow aptitud\_mejorpos_i$

Fin Si

Fin Para

Para i = 1 hasta tamaño (*Nube*) **hacer**

Calcular la velocidad $\boldsymbol{v}_i$ de $\boldsymbol{x}_i$, en base a los valores

$\boldsymbol{x_i}$, $\boldsymbol{mejorpos_i}$ y $\boldsymbol{mejorpos}$

Calcular la nueva posición de $\boldsymbol{x}_i$, de su valor actual y $\boldsymbol{v}_i$

Fin Para

Fin Mientras

Salida: Devuelve la mejor solución encontrada.

Las variantes a este algoritmo dependen fundamentalmente de:

- La implementación de las partículas.
- Cálculo o actualización del vector velocidad.
- Cálculo o actualización del movimiento de las partículas.

- Representación de las soluciones.

A continuación se describen los pseudocódigos del PSO para problemas con codificación continua y con codificación binaria.

3.5.2 Pseudocódigo del PSO para codificación continua

El algoritmo PSO para codificación continua es la versión original y más utilizada, concebida en espacios de búsqueda con variables continuas. En el siguiente pseudocódigo ya se desarrolla la actualización de las velocidades de las partículas y sus nuevas posiciones.

ALGORITMO 2 – PSO PARA CODIFICACIÓN CONTINUA

$t = 0$

Nube ← Inicializar Nube de Partículas

Mientras no se alcance la condición de parada **hacer**

$t = t + 1$

Para i = 1 hasta tamaño (*Nube*) **hacer**

Evaluar cada partícula $\boldsymbol{x}_i$ de la *Nube*

Si $aptitud\_x_i$ es mejor que $aptitud\_mejorpos_i$ **entonces**

$mejorpos_i \leftarrow x_i$;

$aptitud\_mejorpos_i \leftarrow aptitud\_x_i$

Fin Si

Si $aptitud\_mejorpos_i$ es mejor que *aptitud_mejorpos* **entonces**

mejorpos ← $mejorpos_i$;

aptitud_mejorpos ← $aptitud\_mejorpos_i$

Fin Si

Fin Para

Para i = 1 hasta tamaño (*Nube*) **hacer**

$$\mathbf{v}_i^t \leftarrow \omega^{t-1} \cdot \mathbf{v}_i^{t-1} + \varphi_1 \cdot rand_1 \cdot (\mathbf{mejorpos}_i - \mathbf{x}_i^{t-1}) + \varphi_2 \cdot rand_2 \cdot (\mathbf{mejorpos} - \mathbf{x}_i^{t-1})$$

$\mathbf{x}_i^t \leftarrow \mathbf{x}_i^{t-1} + \mathbf{v}_i^t$

Fin Para

Fin Mientras

Salida: Devuelve la mejor solución encontrada.

Este pseudocódigo es válido tanto para el PSO Local como PSO Global. Ambos se distinguen según la definición de la mejor partícula del vecindario, ***mejorpos***, así:

- En PSO Local:
  mejorpos = $\boldsymbol{mejorposl_i}$ ≡ Posición de la mejor partícula perteneciente al entorno local, es decir, a las partículas inmediatamente cercanas a $\boldsymbol{x_i}$.

- EN PSO Global:
  mejorpos =***mejorposg*** ≡ Posición de la mejor partícula considerando la nube completa.

3.5.3 Pseudocódigo del PSO para codificación binaria

Existen muchos problemas de optimización que se plantean con espacios de búsqueda discretos. La resolución de estos problemas con algoritmos PSO requiere que sus partículas se representen de una forma que se adapte al espacio discreto. La codificación binaria es válida para afrontar la representación de las soluciones y la optimización del problema con garantías de éxito.

En un algoritmo con este tipo de codificación, las posiciones de las partículas en el espacio de búsqueda se representan mediante cadenas de bits, por lo que el algoritmo, y, sobre todo, los operadores, pueden variar respecto al PSO en versión continua. El movimiento consiste en el cambio de valor de algunos de los bits que representan a la partícula.

El primer algoritmo PSO binario fue propuesto por Kennedy y Eberhart en [96]. Estos autores representan la velocidad mediante un vector real. Su actualización viene dada por:

$$\mathbf{v}_i^t \leftarrow \omega^{t-1} \cdot \mathbf{v}_i^{t-1} + \varphi_1 \cdot rand_1 \cdot (\mathbf{mejorpos}_i - \mathbf{x}_i^{t-1}) + \varphi_2 \cdot rand_2 \cdot (\mathbf{mejorposg} - \mathbf{x}_i^{t-1}) \quad (3.4)$$

La velocidad del bit *j* de la partícula *i*, $v_{i,j}$, se interpreta como la probabilidad de que tome el valor de '1'. Si la velocidad es alta con respecto a un determinado valor umbral, el nuevo valor será 1, y si es baja tomará el valor de 0. El valor umbral, ρ, está comprendido en el intervalo [0, 1] y viene dado por la siguiente expresión:

Si $\rho_i^t < sig(v_{i,j}^t)$ entonces $x_{i,j}^t = 1$; para otro caso $x_{i,j}^t = 0$

Donde *sig*( ) es la función sigmoidal. Se emplea para transformar el valor de la velocidad dentro del rango del valor umbral establecido [0, 1]. La función sigmoidal viene dada por la siguiente ecuación:

$$sig(v_{i,j}^t) = \frac{1}{1 + \exp\{-v_{i,j}^t\}} \quad (3.5)$$

Por tanto, para cada iteración y partícula se obtiene una cadena completa de bits con valores de 0 ó 1.

ALGORITMO 3 – PSO BINARIO - VERSION KENNEDY Y EBERHART

$t = 0$

Nube ← Inicializar Nube de Partículas

Mientras no se alcance la condición de parada **hacer**

$t = t + 1$

Para i = 1 hasta tamaño (*Nube*) **hacer**

Evaluar cada partícula $\boldsymbol{x}_i$ de la *Nube*

Si $aptitud\_x_i$ es mejor que $aptitud\_mejorpos_i$ **entonces**

$mejorpos_i \leftarrow x_i$;

$aptitud\_mejorpos_i \leftarrow aptitud\_x_i$

Fin Si

Si $aptitud\_mejorpos_i$ es mejor que $aptitud\_mejorposg$ **entonces**

$mejorposg \leftarrow mejorpos_i$;

$aptitud\_mejorposg \leftarrow aptitud\_mejorpos_i$

Fin Si

Fin Para

Para i = 1 hasta tamaño (*Nube*) **hacer**

$$\mathbf{v}_i^t \leftarrow \omega^{t-1} \cdot \mathbf{v}_i^{t-1} + \varphi_1 \cdot rand_1 \cdot (\mathbf{mejorpos}_i - \mathbf{x}_i^{t-1}) + \varphi_2 \cdot rand_2 \cdot (\mathbf{mejorposg} - \mathbf{x}_i^{t-1})$$

Si $\rho_i^t < sig(v_{i,j}^t)$ **entonces** $x_{i,j}^t = 1$; **sino** $x_{i,j}^t = 0$

Fin Para

Fin Mientras

Salida: Devuelve la mejor solución encontrada.

Esta primera versión binaria convierte toda la información de la dirección que lleva la partícula en dos únicos niveles de decisión, limitados por el valor umbral. En muchos casos, esta limitación provoca la pérdida de la eficiencia del algoritmo.

En los últimos años, han salido otras versiones que intentan mejorar la eficiencia del PSO binario inicialmente propuesto por Kennedy y Eberhart. Así, Afshinmanesh y sus colaboradores proponen una versión diferente en [2].

El algoritmo de Afshinmanesh y col. utiliza conceptos como la distancia de Hamming y los operadores lógicos AND (·), OR (+) y XOR ($\oplus$). La distancia de Hamming, *dH*, se define como el número de bits que tienen que cambiarse para transformar una palabra de código válida en otra también válida.

Si dos palabras o cadenas de bits se diferencian en una distancia de Hamming, *dH*, se necesitan *dH* errores para trasformarse una en la otra. Las ecuaciones que definen la actualización de la posición de las partículas y la velocidad son las siguientes:

$$x_{i,j}^{t} = x_{i,j}^{t-1} \oplus v_{i,j}^{t} \tag{3.6}$$

$$v_{i,j}^{t} = c_{1i,j} \cdot dH_{1i,j}^{t-1} + c_{2i,j} \cdot dH_{2i,j}^{t-1} \tag{3.7}$$

Donde:

- $x_{i,j}^{t} \equiv$ Valor del bit en la posición *j* del vector de posición de la partícula *i*, $\mathbf{x}_i = (x_{i,1}, x_{i,2}, ..., x_{i,N})$ en el momento o iteración *t*.
- $v_{i,j}^{t} \equiv$ Valor del bit en la posición *j* del vector de velocidad *i*, $\mathbf{v}_i = (v_{i,1}, v_{i,2}, ..., v_{i,N})$ en la iteración *t*.
- $c_{1i,j} \equiv$ Valor binario aleatorio perteneciente al vector $\mathbf{c}_{1i} = (c_{1i,1}, c_{1i,2}, ..., c_{1i,N})$.
- $c_{2i,j} \equiv$ Valor binario aleatorio perteneciente al vector $\mathbf{c}_{2i} = (c_{2i,1}, c_{2i,2}, ..., c_{2i,N})$.
- $dH_{1i,j}^{t} \equiv$ Es el bit en la posición *j* del vector $\mathbf{dH}_{1i} = (dH_{1i,1}, dH_{1i,2}, ..., dH_{1i,N})$ que denota la distancia de Hamming entre la posición de la partícula $\mathbf{x}_i = (x_{i,1}, x_{i,2}, ..., x_{i,N})$ y su mejor posición, $\mathbf{mejorpos}_i = (mejorpos_{i,1}, mejorpos_{i,2}, ..., mejorpos_{i,N})$ en la iteración *t.*
- $dH_{2i,j}^{t} \equiv$ Es el bit en la posición *j* del vector $\mathbf{dH}_{2i} = (dH_{2i,1}, dH_{2i,2}, ..., dH_{2i,N})$ que denota la distancia de Hamming entre la posición de la partícula $\mathbf{x}_i = (x_{i,1}, x_{i,2}, ..., x_{i,N})$ y la mejor posición, $\mathbf{mejorposg} = (mejorposg_1, mejorposg_2, ..., mejorposg_N)$ en la iteración *t*.

El cálculo de la distancia de Hamming se realiza mediante las siguientes ecuaciones:

$$dH_{1i,j} = mejorpos_{i,j} \oplus x_{i,j} \tag{3.8}$$

$$dH_{2i,j} = mejorposg_{j} \oplus x_{i,j} \tag{3.9}$$

ALGORITMO 4 – PSO BINARIO – VERSION AFSHINMANESH

$t = 0$

Nube ← Inicializar Nube de Partículas

Mientras no se alcance la condición de parada **hacer**

$t = t + 1$

Para i = 1 hasta tamaño (*Nube*) **hacer**

Evaluar cada partícula $\mathbf{x}_i$ de la *Nube*

Si $aptitud\_x_i$ es mejor que $aptitud\_mejorpos_i$ **entonces**

$mejorpos_i \leftarrow x_i$;

$aptitud\_mejorpos_i \leftarrow aptitud\_x_i$

Fin Si

Si $aptitud\_mejorpos_i$ es mejor que $aptitud\_mejorposg$ **entonces**

$mejorposg \leftarrow mejorpos_i$;

$aptitud\_mejorposg \leftarrow aptitud\_mejorpos_i$

Fin Si

Fin Para

Para i = 1 hasta tamaño (*Nube*) **hacer**

$$\mathbf{v}_i^t = \mathbf{c}_{1i} \cdot \mathbf{dH}_{1i}^{t-1} + \mathbf{c}_{2i} \cdot \mathbf{dH}_{2i}^{t-1}$$

$$\mathbf{x}_i^t = \mathbf{x}_i^{t-1} \oplus \mathbf{v}_i^t$$

Fin Para

Fin Mientras

Salida: Devuelve la mejor solución encontrada.

En las publicaciones existentes se pueden encontrar otros modelos del algoritmo PSO para binario adaptados a problemas o para aplicaciones específicas, tales como los planteados en [5, 141, 146].