Universidad de La Habana
Facultad de Matemática y Computación

Estrategias para mejorar el Balance entre Exploración y Explotación en Optimización de Enjambre de Partículas

Autor:
Lic. Yenny Noa Vargas

Tutores:
Dr. Stephen Chen
Dr. Juan Manuel Otero

Tesis presentada para obtener el
Título de Máster en Ciencias Matemáticas
Mención Optimización

Diciembre de 2011

A mis padres

Resumen

La explotación y exploración del espacio de búsqueda son factores claves que deben considerarse para el buen desempeño de cualquier técnica de búsqueda y optimización global. Mientras la explotación guía la búsqueda teniendo en cuenta la información que se obtiene de las mejores soluciones encontradas hasta el momento, la exploración propicia descubrir regiones sin explorar y evitar una convergencia prematura. Lograr un balance entre estos dos objetivos es un problema de vital importancia que enfrenta la mayoría de las técnicas de búsqueda y optimización actuales.

En este trabajo se proponen nuevas estrategias para mejorar el balance entre la exploración y explotación en la metaheurística Optimización de Enjambre de Partículas. El análisis de sus efectos en dos versiones del algoritmo estándar complementa la base de conocimiento que se tiene sobre su funcionamiento y puede ser aprovechado en investigaciones futuras que tengan como objetivo perfeccionar su diseño y mejorar su rendimiento.

Abstract

Exploration and exploitation are two important factors to consider in the design of optimization techniques. To perform an effective balance between exploration and exploitation, two tasks must be taken into account: quickly identify regions in the search space with high quality solutions without wasting too much time in regions which are either already explored or which do not provide high quality solutions, and perform an intense search exploiting the collected search experience to locate the optimal solutions. These two tasks are conflicting and equally important so a trade-off between these two objectives – exploration and exploitation – must be achieved.

In this paper, new strategies are introduced for improving the balance between exploration and exploitation in Particle Swarm Optimization. The analysis of these strategies in two versions of the standard algorithm, helps to better understand this metaheuristic and provide some insights that can be exploited in future studies that aim to improve their design and performance.

Índice general

Índice de figuras

Índice de tablas

Introducción

De las palabras Griegas *heuriskein* y *meta*, que significan "buscar" y "en un nivel superior", proviene la palabra *metaheurística*. El término fue introducido por Fred Glover en un artículo sobre Búsqueda Tabú [1] con el objetivo de crear una alternativa para resolver problemas de optimización complejos, que no podían ser resueltos con los métodos clásicos de optimización existentes en un tiempo computacional razonable.

Aunque no existe una definición estándar de metaheurística, la mayoría de los investigadores se refieren a ella como un procedimiento iterativo de alto nivel, que guía y modifica las operaciones de una heurística subordinada mediante la combinación de estrategias inteligentes para explorar y explotar eficientemente el espacio de búsqueda, evitar estancamientos en óptimos locales, y encontrar soluciones de gran calidad.

La *exploración* se refiere a la tarea de examinar eficientemente toda región del espacio de búsqueda y encontrar nuevas soluciones potencialmente mejores. Por otro lado, la *explotación* es el proceso de mejorar y combinar (intensificar) los rasgos de las mejores soluciones encontradas durante el proceso de exploración, con el objetivo de encontrar soluciones de mayor calidad.

Los algoritmos que favorecen la explotación poseen una mayor velocidad de convergencia, pero a riesgo de quedar atrapados en un óptimo local. Aquellos que exploran en exceso pueden no llegar a converger a la solución óptima o hacerlo después de pasado demasiado tiempo, de manera accidental. En este contexto, estos dos objetivos – exploración y explotación – son opuestos aunque igualmente importantes, de ahí que se considere una tarea fundamental hallar un balance adecuado entre ellos al diseñar cualquier técnica de búsqueda y optimización.

Estrategias para balancear la exploración y explotación se pueden encontrar en muchas de las metaheurísticas existentes. Por ejemplo, en Búsqueda en Vecindades Variables [2, 3] se definen diferentes estructuras de vecindades,

las cuales se van intercambiando en dependencia del estado de la búsqueda. Este algoritmo funciona bajo la hipótesis de que soluciones que son óptimos locales respecto a una estructura de vecindad, no necesariamente lo serán respecto a otra. De esta forma, ante una convergencia prematura del algoritmo en un óptimo local, el cambio de estructura de vecindad puede significar una vía de escape y por tanto la posibilidad de encontrar una mejor solución. En Búsqueda Tabú [4, 5] se hace uso de una memoria a corto plazo (*lista tabú*) para almacenar la trayectoria de la búsqueda. Esta información es utilizada para realizar búsquedas exhaustivas alrededor de las mejores soluciones encontradas y promover la exploración de soluciones con características similares a estas (intensificación), así como para enfatizar la exploración de regiones no visitadas o de soluciones que difieren de una manera significativa de aquellas que ya han sido examinadas (diversificación)[1]. En Recocido Simulado [7, 8], la estrategia consiste en el empleo de un parámetro (*temperatura*) para determinar la probabilidad de aceptar soluciones peores a la solución actual. Generalmente se comienza con un valor alto favoreciendo la exploración de todo el espacio de búsqueda y se va disminuyendo su valor, gradualmente, hasta comportarse como un algoritmo de búsqueda local simple.

Todos los métodos descritos anteriormente se clasifican como *métodos de trayectorias* (trajectory methods) porque manejan una única solución en cada iteración del proceso de búsqueda y el paso de una solución a otra en cada iteración va describiendo una trayectoria en el espacio de búsqueda. Pueden verse como extensiones "inteligentes" de una búsqueda local, pues incorporan estrategias que tienen como objetivo principal escapar de los óptimos locales para continuar la exploración de otras regiones en busca del óptimo global.

Existe otro grupo de metaheurísticas cuyo proceso de búsqueda describe la evolución de un conjunto de puntos en el espacio. Los métodos que en cada iteración manejan un conjunto de soluciones en lugar de una sola se conocen como *métodos basados en población* (population-based methods). A este grupo pertenecen los Algoritmos Evolutivos y los algoritmos de Inteligencia de Enjambres.

Los Algoritmos Evolutivos, entre los que se encuentran los Algoritmos Genéticos [9, 10] y las Estrategias Evolutivas [11], pueden describirse como una carrera por la supervivencia, donde las soluciones más aptas (de mejor

[1]Intensificación y diversificación son términos introducidos por Glover y Laguna [6] para referirse a conceptos similares a explotación y exploración. En este trabajo se usarán indistintamente ambas definiciones.

evaluación) tendrán más posibilidades de sobrevivir y, por tanto, de pasar sus características a las nuevas generaciones por medio de la reproducción y la herencia. Simulando el proceso de la evolución de las especies y la selección natural descrito por Darwin, estos algoritmos se componen de operadores de cruzamiento, mutación y selección, entre otros.

La Inteligencia de Enjambre [12] se basa en el comportamiento colectivo de sistemas biológicos auto-organizados y descentralizados, tales como, las colonias de hormigas, manchas de peces, bandadas de aves, etc. Los algoritmos que pertenecen a esta clase se componen de agentes simples, que interactúan entre ellos y con su ambiente siguiendo reglas simples, que conducen a un comportamiento global complejo, a pesar de la ausencia de un mecanismo de control central. Ejemplos de algoritmos en esta clase son la Optimización de Colonia de Hormigas [13] y la Optimización de Enjambre de Partículas [14].

El hecho de manipular un conjunto de soluciones favorece a estos algoritmos con una manera natural de recorrer (explorar) el espacio de búsqueda. Una dificultad que presentan los mismos es su capacidad para mantener la diversidad y evitar una convergencia prematura.

En los Algoritmos Genéticos, por ejemplo, este problema se enfrenta por medio de los operadores genéticos. De la intensificación de la búsqueda (explotación) se hace cargo el operador de selección, la creación de nuevas soluciones (exploración) se realiza por medio de la mutación, aunque se han diseñado otras estrategias con el afán de mantener la diversidad (e.g crowding strategy [15], niching and fitness sharing [16]). El operador de cruzamiento puede funcionar como intensificador y diversificador de la búsqueda en dependencia del estado de la misma y la representación de las soluciones. En Optimización de Colonia de Hormigas, la combinación de los mecanismos para depositar y evaporar la feromona constituye una manera de equilibrar la explotación de rasgos existentes en soluciones examinadas y la exploración de otros nuevos.

Lograr un balance adecuado entre explotación y exploración en las metaheurísticas y, en general, de cualquier técnica de búsqueda y optimización, ha sido tema de estudio por muchos años; y su importancia ha sido reflejada por diferentes investigadores:

> *"Después de que un óptimo local sea encontrado, todos los puntos en la región que son atraídos por él pierden interés para la optimización. Se debería evitar desperdiciar demasiado tiempo de cómputo en esta región solamente y la diversificación debería ser activada. Por otro lado, asumiendo que existe una relación entre*

> *los puntos vecinos y sus costos de función, debería invertirse un poco de esfuerzo en la búsqueda de mejores puntos situados cerca del óptimo local recién encontrado (intensificación). Los dos requisitos son contradictorios y la búsqueda de un equilibrio adecuado de la diversificación y la intensificación es un tema crucial en las heurísticas."* [17]

> *"Una metaheurística tendrá éxito en un problema de optimización dado si logra establecer un equilibrio entre la explotación de la experiencia acumulada durante la búsqueda y la exploración del espacio de búsqueda para identificar las regiones con las soluciones de alta calidad."* [18]

Caso de estudio y objetivos

La Optimización de Enjambre de Partículas es una metaheurística de reglas muy simples que ha mostrado excelentes habilidades para la búsqueda de soluciones óptimas en diversos problemas de optimización. Sin embargo, al igual que muchas otras metaheurísticas, enfrenta el problema de la posible convergencia prematura en regiones de óptimos locales. Es por ello que, desde su invención, se han propuesto diversas estrategias para mejorar el balance entre exploración y explotación y así perfeccionar su rendimiento [19, 20, 21, 22]. Como continuación a estos trabajos, el objetivo general de esta investigación es:

- Proponer nuevas estrategias para mejorar el balance entre la exploración y explotación de la metaheurística Optimización de Enjambre de Partículas.

En función de este objetivo general se establecieron los siguientes objetivos específicos:

1. Investigar los antecedentes, conceptos básicos de la Optimización de Enjambre de Partículas y sus variantes, haciendo énfasis en las técnicas propuestas para mejorar el balance entre exploración y explotación.

2. Proponer nuevas estrategias para mejorar el balance entre exploración y explotación, y como consecuencia, el rendimiento del algoritmo.

3. Implementar dichas estrategias y evaluarlas sobre un conjunto de problemas con diferentes características.

4. Efectuar un análisis de los resultados alcanzados, que permita arribar a conclusiones respecto al funcionamiento de la Optimización de Enjambre de Partículas y a la efectividad y posibles aplicaciones de las estrategias que se propongan.

Este documento se estructura en cuatro capítulos. En el primero se exponen los antecedentes, conceptos básicos y la versión original de la Optimización de Enjambre de Partículas; las principales modificaciones realizadas al algoritmo, así como la nueva definición de su versión estándar; y finalmente, haciendo énfasis en las estrategias propuestas para mejorar el balance entre exploración y explotación, algunas de sus variantes más recientes.

En el segundo capítulo, motivado por el estudio de una de estas variantes, se introduce una estrategia de búsqueda que, apoyándose en la noción de Distancia Manhattan, modifica la trayectoria de las partículas y reduce el número de dimensiones analizadas durante el proceso de búsqueda del algoritmo.

En el tercer capítulo se introducen dos nuevas estrategias. La primera se inspira en el proceso de intensificación de metaheurísticas como Búsqueda Tabú, mientras que la segunda se inserta como una estrategia de diversificación, cuya efectividad se debe a la prolongación de la fase de exploración en el algoritmo.

Al final de los capítulos segundo y tercero, luego de exponer los problemas que son objeto de estudio durante el proceso de experimentación y la configuración de parámetros utilizados por los algoritmos implementados, se analizan los efectos que producen las estrategias propuestas a la Optimización de Enjambres de Partículas.

Finalmente, antes de mostrar las referencias bibliográficas consultadas y los anexos al trabajo, se presentan las conclusiones obtenidas durante la investigación realizada y las recomendaciones para trabajos futuros.

Capítulo 1

Fundamentos de la Optimización de Enjambre de Partículas

La Optimización de Enjambre de Partículas – PSO por sus siglas en inglés – junto a la Optimización de Colonia de Hormigas y Optimización de Colonia de Abejas, se incluye dentro de lo que hoy se conoce como Inteligencia de Enjambre [12, 23]. Este es el campo de investigación, dentro de la Inteligencia Artificial, que estudia el comportamiento de los enjambres en la naturaleza. Inspirado en ello, sus algoritmos se conforman por individuos simples que cooperan mediante mecanismos de auto-organización, es decir, sin ningún mecanismo de control central.

PSO se inspira, en particular, en el comportamiento social de organismos biológicos, específicamente, en la habilidad de algunas especies para colaborar, como grupo, en la localización de recursos deseables en un área determinada – bandadas de pájaros y manchas de peces, en busca de alimento y refugio. Este comportamiento se relacionó con la búsqueda de soluciones a ecuaciones no lineales en un espacio de búsqueda real, dando origen así a una nueva técnica de optimización global. Aunque la versión original fue diseñada para resolver problemas de optimización continuos no lineales, se pueden encontrar variantes del algoritmo para resolver problemas de optimización discretos y combinatorios [24, 25, 26].

El objetivo de este capítulo, primeramente, es presentar los antecedentes de PSO, sus conceptos básicos, las principales modificaciones que se le han realizado desde su creación, así como la variante estándar del algoritmo. En una segunda parte, mediante la descripción de variantes de la versión

estándar de PSO propuestas en trabajos precedentes, se presentarán algunas de las técnicas que se utilizan para mejorar el balance entre exploración y explotación del algoritmo, y como consecuencia, su rendimiento.

1.1. Antecedentes

La primera versión del algoritmo PSO [14] se atribuye al ingeniero eléctrico Russell Eberhart y al psicólogo social James Kennedy. En conjunto, hicieron un estudio de las investigaciones realizadas por otros científicos [27, 28] que, atraídos por la estética de las coreografías de las bandadas de pájaros, se interesaron por descubrir las reglas subyacentes que permitían a las aves volar sincrónicamente y cambiar con frecuencia de dirección, dispersándose o reagrupándose. De este estudio surgió la hipótesis fundamental para el desarrollo de PSO:

> *"...el intercambio social de información entre individuos de una misma especie ofrece una ventaja evolutiva..."* [14]

Kennedy y Eberhart comenzaron por simular un ambiente social simplificado. El experimento original tuvo como objetivo representar gráficamente la gracia y lo imprevisible de las coreografías de las bandadas de pájaros. Para esta primera simulación, que llamaron *Nearest Neighbor Velocity Matching and Craziness*, crearon una población de agentes, asignándoles posiciones (px, py) y velocidades (vx, vy) aleatorias dentro de una matriz de dos dimensiones. En cada iteración, a cada agente se le asignaba la velocidad del agente vecino más cercano. Esta regla muy simple dio como resultado un movimiento sincrónico, sin embargo, al poco tiempo los agentes se aglomeraban alrededor de una misma posición y no cambiaban de dirección. De ahí que decidieran incorporar a la simulación una variable aleatoria que llamaron *craziness*, la cual se añadía a componentes vx y vy seleccionadas al azar en cada iteración. Esto introdujo suficiente variación al sistema obteniéndose una simulación más realista.

Asociando la optimización de funciones con la atracción natural de grupos de animales por ciertas zonas geográficas, como puede ser la rama de un árbol ó un comedero para una bandada de aves y un campo sembrado de flores para un enjambre de abejas, Kennedy y Eberhart realizaron una segunda simulación. La llamaron *The Cornfield Vector*, pues definieron un punto de atracción – *cornfield* – representado por un vector de dos dimensiones en la matriz y cada agente estaba programado para evaluar su posición actual

(px, py) en términos de la ecuación:

$$Eval = \sqrt{(px-100)^2} + \sqrt{(py-100)^2} \tag{1.1}$$

de modo que el valor en la posición $(100, 100)$ fuera cero.

En la simulación cada agente "recordaba" en qué posición había alcanzado su mejor valor. Denominaron a este valor *pbest* siendo $pbestx_i$ y $pbesty_i$ las coordenadas de la mejor posición alcanzada por el agente i. También cada agente "sabía" cual era la mejor posición encontrada por cualquiera de los otros. Denominaron a este valor *gbest* siendo *gbestx* y *gbesty* las coordenadas del agente con la mejor posición de todas.

El movimiento de los agentes se controlaba en términos de *pbest* y *gbest*. La velocidad del agente i se actualizaba primero teniendo en cuenta la posición actual (px_i, py_i) respecto a su *pbest*:

$$vx_i = \begin{cases} vx_i - rand() * c_1 & \text{si} \quad px_i > pbestx_i \\ vx_i + rand() * c_1 & \text{si} \quad px_i < pbestx_i \end{cases} \tag{1.2}$$

Luego se actualizaba respecto a *gbest* de acuerdo a:

$$vx_i = \begin{cases} vx_i - rand() * c_2 & \text{si} \quad px_i > gbestx \\ vx_i + rand() * c_2 & \text{si} \quad px_i < gbestx \end{cases} \tag{1.3}$$

De manera similar se hacía para la segunda componente vy, primero respecto a *pbest* y luego a *gbest*. Siendo $rand()$ una función que genera números aleatorios con distribución uniforme $(0, 1)$ y c_1, c_2 parámetros constantes del sistema, $rand() * c_1$ y $rand() * c_2$ consituían la mangnitud de la fuerza de atracción que ejercían las posiciones *pbest* y *gbest* sobre la posición actual del agente.

En la simulación, la posición $(100, 100)$ fue marcada con un círculo y los agentes fueron coloreados de modo que se podía verles moverse hasta pararse en la posición objetivo. El resultado de la simulación les sorprendió, pues para valores muy altos de c_1 y c_2 observaron cómo los agentes eran aspirados violentamente por el punto de atracción, mientras que, para valores pequeños, los agentes se agrupaban alrededor del objetivo, y, poco a poco, con sincronía y de una forma más realista, se iban acercando en subgrupos hasta alcanzar finalmente el objetivo.

La variable *pbest* simulaba una memoria autobiográfica y su asociación con la actualización de las velocidades la denominaron "simple nostalgia", en el sentido de que los agentes tendían a retornar al lugar que más les satisfizo en el pasado. Por otro lado, *gbest* representaba un conocimiento público,

algo así como una norma o un estándar que todo agente debía seguir. Con las variables c_1 y c_2 notaron que podían controlar el comportamiento de los agentes y la convergencia del algoritmo: valores muy superiores de c_1 respecto a c_2 producían una excesiva exploración individual de los agentes, mientras que con valores muy superiores de c_2 respecto a c_1, la convergencia era segura pero prematura.

Las simulaciones realizadas revelaron resultados prometedores, pero hasta ese momento con aplicación a la optimización de funciones lineales y de dos dimensiones. Como muchos de los problemas prácticos e interesantes eran no lineales y multidimensionales, extrapolaron las ideas anteriores a matrices de $N \times D$, siendo N el número de agentes y D el número de dimensiones.

Otras simulaciones incluyeron la actualización de las velocidades (*Acceleration by Distance*) a partir de la diferencia existente entre las posiciones actuales y las mejores posiciones encontradas, en lugar de verificar solo el signo de las inecuaciones en las ecuaciones (1.2) y (1.3). Los parámetros c_1 y c_2 fueron sustituídos por la constante 2 lo cual produjo mejores resultados aunque dejaron como tema futuro la investigación de su valor óptimo. En la versión simplificada de estas simulaciones (*Current Simplified Version*) los agentes actualizaban sus velocidades mediante la ecuación:

$$v_{i,j} = v_{i,j} + 2 * rand() * (pbest_{i,j} - p_{i,j}) + 2 * rand() * (gbest_j - p_{i,j}) \quad (1.4)$$

donde el subíndice i, j denotaba el agente i y la dimensión j.

En concordancia con el artículo [29], donde se desarrolla un modelo para aplicaciones sobre sistemas de vida artificial y se formulan cinco principios básicos de la Inteligencia de Enjambre, los autores deciden usar el término *enjambre* para referirse a la población de agentes, pues por sus características, la nueva técnica de optimización se adhería a estos cinco principios. Por otro lado, asumieron como convenio el término *partícula* para referirse a los agentes, considerando que poseían velocidades y aceleraciones aunque carecieran de masa y volumen.

De manera general, las simulaciones y aspectos anteriores constituyeron el origen de la versión canónica de PSO, que se describe a continuación.

1.2. Versión canónica de PSO

Sean $f : \mathbb{R}^n \to \mathbb{R}$ la función que se desea minimizar[1] y S la cantidad de partículas que componen el enjambre. Se definen para cada partícula cuatro vectores de dimensión n como atributos: x_i, v_i, $pbest_i$ y $gbest_i$. La posición x_i representa una solución potencial para la función objetivo, la velocidad v_i representa la dirección e intensidad del movimiento de la partícula, $pbest_i$ representa la mejor posición encontrada individualmente y $gbest_i$ la mejor posición encontrada por las partículas en su vecindad hasta el momento actual [30].

En el algoritmo 1 se describen los pasos de la versión canónica de PSO. Luego de la inicialización de sus atributos, cada partícula procede a recorrer el espacio de búsqueda mediante la actualización de su velocidad y posición. Este proceso ocurre de manera iterativa y culmina luego de trascurrido cierto número prefijado de iteraciones T.

Algoritmo 1 PSO original

1. Inicializar constantes y variables (T, S, c_1, c_2, x_i^0, v_i^0)
2. **for** $i = 1$ to S **do**
3. $\quad pbest_i^0 \leftarrow x_i^0$
4. **end for**
5. **for** $i = 1$ to S **do**
6. $\quad$ Actualizar $gbest_i^0$
7. **end for**
8. **for** $t = 1$ to T **do**
9. $\quad$ **for** $i = 1$ to S **do**
10. $\quad\quad$ Actualizar v_i^t y x_i^t
11. $\quad\quad$ Evaluar $f(x_i^t)$ y actualizar $pbest_i^t$
12. $\quad$ **end for**
13. $\quad$ **for** $i = 1$ to S **do**
14. $\quad\quad$ Actualizar $gbest_i^t$
15. $\quad$ **end for**
16. **end for**
17. **return** $gbest \leftarrow min_{gbest_i}\left\{f(gbest_i^T)\right\}$

Las velocidades y posiciones se actualizan mediante las ecuaciones:

$$v_{i,j}^{t+1} = \underbrace{v_{i,j}^t} + \underbrace{c_1 r_{1_{i,j}}^t [pbest_{i,j}^t - x_{i,j}^t]} + \underbrace{c_2 r_{2_{i,j}}^t [gbest_{i,j}^t - x_{i,j}^t]} \tag{1.5}$$

$$x_{i,j}^{t+1} = x_{i,j}^t + v_{i,j}^{t+1} \tag{1.6}$$

[1]Sin pérdida de generalidad se puede considerar solamente el problema de minimizar una función $f(x)$, pues si el problema es de maximizar, puede formularse mediante el problema equivalente de minimizar $-f(x)$.

donde para toda partícula i: $v_{i,j}$ y $x_{i,j}$ representan la dimensión j-ésima de su velocidad y posición respectivamente; $r_{1_{i,j}}$ y $r_{2_{i,j}}$ son elementos de dos secuencias generadas aleatoriamente con distribución uniforme en el rango $(0,1)$; c_1 y c_2 son las constantes de aceleración que deciden la manera en que influirán las posiciones *pbest* y *gbest*, respectivamente, en el movimiento de las partículas y t es el número de iteraciones completadas. El uso de la variable t como superíndice en las ecuaciones denota el valor de cada una de las variables en la iteración t.

Como se evidencia en las ecuaciones (1.5) y (1.6), la dirección e intensidad de los movimientos, realizados por cada partícula en el espacio de búsqueda, está determinada por la influencia de tres componentes. El primero, es el *impulso o ímpetu* que representa la fuerza que se ejerce sobre la partícula para que continúe la dirección que lleva en el momento actual. El segundo, es el *componente cognitivo* que representa la fuerza que surge a partir de la atracción de la partícula por su *pbest*, y el tercero, es el *componente social* que representa la fuerza que surge a partir de la atracción de la partícula por el *gbest* de su vecindad [31].

Una mayor influencia del componente cognitivo en la ecuación de la velocidad trae como consecuencia que las partículas realicen una búsqueda local basada en su experiencia individual. En cambio, si la influencia del componente social es mayor, entonces se favorece la exploración de otras áreas descubiertas por las partículas en su vecindad. La Figura 1.1 muestra un ejemplo de actualización de la velocidad y la posición de una partícula i en un espacio de dos dimensiones en la iteración t. Se destacan con diferentes colores cada una de estas componentes.

Un aspecto que caracteriza y favorece el proceso de optimización en PSO es la actualización de la experiencia personal de cada partícula (*pbest*), así como el intercambio de información que se produce entre ellas a través de la actualización de *gbest*. La mejor posición de una partícula es actualizada si su posición actual es mejor que aquellas por las que ha transitado hasta el momento y se realiza mediante la ecuación:

$$pbest_i^{t+1} = \begin{cases} pbest_i^t & \text{si} \quad f(x_i^{t+1}) \geq f(pbest_i^t) \\ x_i^{t+1} & \text{si} \quad f(x_i^{t+1}) < f(pbest_i^t) \end{cases} \tag{1.7}$$

La actualización de *gbest* para una partícula i se realiza mediante la ecuación (1.8), comparando las mejores posiciones de todas las partículas en su vecindad, usualmente representada por un conjunto V_i que contiene los índices de las partículas que son sus vecinas. En la sección 1.3.3 se definirá el concepto de vecindad y algunas de las topologías de vecindades más cono-

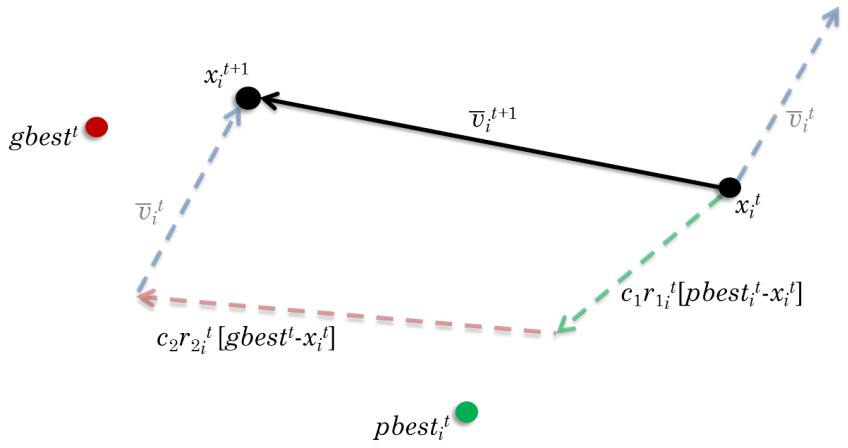


Figura 1.1: Actualización de la velocidad y posición de una partícula en un espacio de dos dimensiones.

cidas.

$$gbest_i^{t+1} = min_{pbest_j} \left\{ f(pbest_j^{t+1}) \right\}, \qquad j \in V_i \tag{1.8}$$

Según las descripciones consultadas de la versión canónica de PSO, el proceso de búsqueda del algoritmo llega a su fin si se alcanza el número de iteraciones previamente definido (T). Luego, se retorna la mejor solución encontrada. No obstante, se pudieran definir otras condiciones de parada, por ejemplo, la mejor solución obtenida no varía durante k iteraciones consecutivas, el enjambre ha convergido a una misma región del espacio o la evaluación de la mejor solución obtenida es menor que un valor prefijado.

1.3. Variaciones y consideraciones

A continuación se describirán algunas de las modificaciones que se han propuesto a la versión canónica de PSO en busca de mejorar su rendimiento. Se presta atención especial a aquellas relacionadas con mejorar el balance entre la exploración y explotación del espacio de búsqueda. Se analizan también algunas consideraciones realizadas respecto al tamaño del enjambre y se define el concepto de topología de vecindad, así como algunas de las topologías de vecindades más utilizadas.

1.3.1. Modificaciones a la ecuación de la velocidad

En la versión canónica de PSO se consideraba una cota *vmax* para limitar el valor máximo de la velocidad de las partículas. Sin esta cota, las velocidades de las partículas y por lo tanto sus posiciones podían incrementarse rápidamente hasta alcanzar el infinito [30].

La utilización de *vmax* para evitar que las velocidades no aumentaran descontroladamente no resultaba aplicable a todo tipo de espacios de búsqueda, y encontrar el valor apropiado para el problema específico que se iba a resolver no era una tarea simple. Entre las modificaciones propuestas a la ecuación de la velocidad se destacan, por sus resultados y vigencia, el *Factor Inercia* (Inertia Weight) y el *Factor de Constricción* (Constriction factor).

Factor Inercia

La propuesta de utilizar un nuevo parámetro w en la ecuación de la velocidad se atribuye a Shi y Eberhart [32] y tenía el objetivo de reemplazar a *vmax*, ajustando la influencia de la velocidad anterior de la partícula en el proceso de búsqueda. Con este parámetro la ecuación para actualizar la velocidad (1.5) quedaba de la siguiente manera:

$$v_{i,j}^{t+1} = \mathbf{w} v_{i,j}^{t} + c_1 r_{1_{i,j}}^{t} [pbest_{i,j}^{t} - x_{i,j}^{t}] + c_2 r_{2_{i,j}}^{t} [gbest_{i,j}^{t} - x_{i,j}^{t}] \tag{1.9}$$

La introducción del parámetro w para controlar la inercia de las partículas resultó ser una mejora y una nueva estrategia para balancear la exploración y la explotación del espacio de búsqueda y eliminó la necesidad de limitar la velocidad con *vmax* en el algoritmo original [30]. Con valores grandes de w (e.g. $> 1,2$) se enfatizaba la exploración global (e.g. diversificando la búsqueda en toda la región del espacio), mientras que, con valores pequeños (e.g. $< 0,8$) se estimula la explotación local (e.g. intensificando la búsqueda en la región actual) [32, 31].

Los autores también sugieren la variación dinámica de w durante el proceso de optimización. Similar a la variación de la temperatura en Recocido Simulado [7, 33], proponen comenzar con un valor mayor que 1, estimulando una temprana exploración del espacio de búsqueda e ir, paulatinamente, disminuyendo su valor hasta uno menor que 1, logrando enfocar el enjambre en la mejor área encontrada durante la fase de exploración. Una implementación de la variante dinámica del factor inercia es la *reducción lineal del factor inercia* (linear reduction of inertia weight) [34] en la que el factor

inercia se actualiza en cada iteración de la siguiente manera:

$$w^{t+1} = \alpha w^t, \qquad 0 < \alpha < 1 \tag{1.10}$$

Los resultados experimentales de esta implementación muestran una mejora en el rendimiento de PSO [34].

Factor de Constricción

Similar al factor inercia, M. Clerc y J. Kennedy [35] proponen la introducción de otro parámetro en la ecuación de la velocidad. El nuevo parámetro χ, conocido como *factor de constricción*, perseguía los mismos objetivos que el parámetro w: modelar el comportamiento del enjambre durante el proceso de búsqueda y controlar la explosión del sistema por el incremento desmesurado de las velocidades de las partículas. La propuesta consistía en obtener χ a partir de las constantes de atracción presentes en la ecuación para actualizar la velocidad usando la siguiente ecuación:

$$\chi = \frac{2}{|\, 2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\,|}, \qquad \varphi = c_1 + c_2 \tag{1.11}$$

La ecuación de la velocidad propuesta con la utilización de este nuevo parámetro fue:

$$v_{i,j}^{t+1} = \chi \left(v_{i,j}^t + c_1 r_{1_{i,j}}^t [pbest_{i,j}^t - x_{i,j}^t] + c_2 r_{2_{i,j}}^t [gbest_{i,j}^t - x_{i,j}^t] \right) \tag{1.12}$$

Para valores de $\varphi < 4$, descubren que el enjambre se acercaba lentamente en espiral a la mejor posición encontrada, pero sin garantía de convergencia, mientras que para valores de $\varphi > 4$ la convergencia era rápida y segura. Aunque los valores de las constantes de atracción c_1 y c_2 son ajustables de modo que puede favorecerse la exploración o la explotación del espacio de soluciones, muchas implementaciones, incluyendo la primera versión del algoritmo, usan el mismo valor para estas dos constantes. En [30] se propone el uso de $\varphi = 4{,}1$ para asegurar la convergencia del algoritmo, obteniéndose los valores $\chi = 0{,}72984$ y $c_1 = c_2 = 2{,}05$. Una prueba de la estabilidad de estos valores puede encontrarse en detalle en [35].

1.3.2. Número de partículas que componen el enjambre

En varias referencias consultadas se analiza la influencia que ejerce el tamaño del enjambre en el rendimiento de PSO. Un incremento del número de partículas puede mejorar las soluciones obtenidas, pero también aumentar

la cantidad de iteraciones y/o el tiempo de ejecución del algoritmo requerido para obtenerlas [31]. En general, para la mayoría de los algoritmos basados en poblaciones (e.g. Algoritmos Evolutivos [9, 11, 10]), se plantea la existencia de un compromiso entre la calidad de las soluciones y el tiempo empleado en la búsqueda [31].

Resultados empíricos muestran que el número de partículas que componen el enjambre influye decisivamente en el rendimiento de PSO dependiendo, por supuesto, del problema a resolver [30]. Algunas funciones de prueba muestran una ligera mejoría en los resultados a medida que el tamaño del enjambre aumenta, mientras que otras, tienden a ser mejor optimizadas con enjambres pequeños. En este sentido, parece ser que no existe un valor óptimo para este parámetro, que permita resolver todo tipo de problemas. No se descarta, por supuesto, los beneficios que pudiera aportar la sintonización de este parámetro para resolver un problema específico.

En el artículo donde se define el nuevo estándar de PSO [30] se reportó que, basado en la experimentación con un conjunto variado de funciones de prueba, ningún enjambre compuesto por entre 20 y 100 partículas produjo resultados claramente superiores (inferiores) para la mayoría de las funciones utilizadas en los experimentos [30]. También, según Talbi [31], la mayoría de las implementaciones de PSO usan enjambres compuestos por entre 20 y 60 partículas.

En general, se recomienda encontrar el tamaño óptimo (o cercano) del enjambre cuando se trata de resolver un problema específico. Si el interés es resolver más de un problema, entonces se aconseja determinar un valor que provea resultados favorables con un ligero margen de diferencia para cada uno de los problemas.

1.3.3. Topología de vecindad o red de comunicación social

Un aspecto necesario e importante a la hora de implementar un PSO es la definición de una *topología de vecindad o red social* para todos los miembros del enjambre. Esto es, el subconjunto de partículas con las cuales cada partícula podrá interactuar durante el proceso de búsqueda.

Formalmente, para cada partícula i debe definirse un V_i con $V_i \subseteq S$, el cual establece que la partícula i podrá comunicarse (intercambiar información) con cualquier partícula $j \in V_i$ mediante la ecuación (1.13). Respecto a la inclusión o exclusión de una partícula en su vecindad se han realizado pruebas que muestran que no existe gran diferencia en el comportamiento

del enjambre si se incluye o no [36].

$$gbest_i = min_{pbest_j} \{f(pbest_j)\}, \qquad j \in V_i \tag{1.13}$$

De esta ecuación se desprende que, si dos partículas i y j comparten la misma vecindad, esto es $V_i = V_j$, entonces $gbest_i = gbest_j$, lo cual significa que estarán siendo atraídas por la misma partícula líder, lo cual podría implicar que terminen explorando la misma región del espacio.

La primera topología de vecindad definida para PSO tenía en cuenta la Distancia Euclidiana, imitando el comportamiento de los modelos biológicos en los que un individuo solo es capaz de comunicarse con otros individuos en una vecindad inmediata [14]. Esta definición de vecindad exigía el establecimiento de un umbral de cercanía e implicaba un alto costo computacional debido al cálculo de las distancias entre todas las partículas en cada iteración, de ahí que comenzaran a utilizarse topologías de vecindades independientes de las posiciones de las partículas.

La primera topología de este tipo propuesta y una de las más utilizadas en la actualidad fue la *topología global*. Conocida también como modelo *gbest*, esta define que todas las partículas están conectadas entre sí, o sea, $V_i = S$ $(\forall i \in S)$, lo cual significa que cada partícula puede transmitir su experiencia a todas las demás y beneficiarse de la de cada una de ellas también.

En el mismo año en que se publica la versión original de PSO [14], sus autores proponen una variante del algoritmo [37] en la que se usa una *topología local*. También conocida como modelo *lbest*, la topología local se refiere a cualquier red de comunicación no global en la que cada partícula intercambia información con un subconjunto propio de todo el enjambre, o sea, $V_i \subset S$ $(\forall i \in S)$.

La forma más común del modelo local es la *topología de anillo*, en la que el enjambre se organiza en formación circular y cada partícula se conecta con las partículas adyacentes. Una manera de definir esta topología para un número variable de vecinos es mediante la fórmula:

$$V_i = \bigcup_{j=i-m}^{i+m} I(j) \quad m = \lfloor \frac{n}{2} \rfloor \tag{1.14}$$

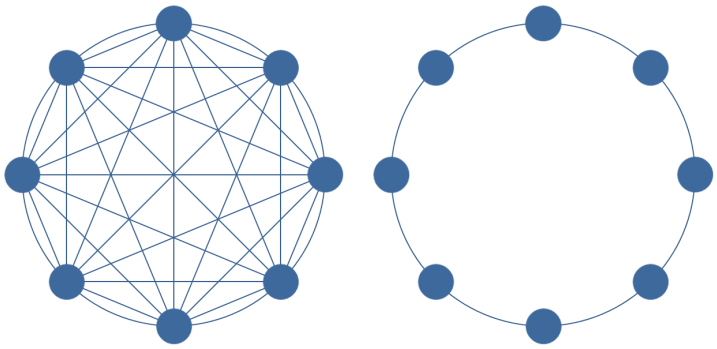
con

$$I(j) = \begin{cases} |S| - j & \text{si} \quad j < 1 \\ j & \text{si} \quad 1 \leq j \leq |S| \\ j - |S| & \text{si} \quad j > |S| \end{cases} \tag{1.15}$$

donde n $(n \leq |S|)$ es la cantidad de partículas vecinas y $|S|$ el tamaño del enjambre. Nótese que cada partícula pertenece a su vecindad, así como las

m partículas adyacentes a ambos lados. Nótese, además, que esta fórmula define vecindades simétricas, o sea, cada partícula posee igual cantidad de partículas vecinas de un lado y del otro [38].

En la figura 1.2 se muestra el modelo lbest para $n = 2$ y el modelo gbest que puede verse como un caso particular del anterior con $n = |S|$. Las partículas han sido representadas por círculos y las conexiones entre ellas mediante arcos y rectas.



(a) Topología de estrella (gbest) (b) Topología de anillo (lbest)

Figura 1.2: Ejemplos de estructuras de vecindades topológicas

Aunque se han definido otras topologías de vecindades [39, 36] los modelos más utilizados en la actualidad son el modelo global y el modelo local en su variante de anillo con $n = 2$. En lo adelante, cuando se mencione al modelo local (lbest) se estará haciendo referencia a esta variante en particular.

Modelo *gbest* vs Modelo *lbest*

Existe una relación directa entre el tamaño de las vecindades de las partículas y la velocidad de convergencia del enjambre. Para grandes vecindades la mejor posición encontrada será usada por mayor cantidad de partículas para actualizar sus posiciones, de modo que la variación del enjambre de una iteración a otra será mucho mayor. En cambio, con vecindades pequeñas, la mejor posición encontrada se propagará más lentamente y el cambio en las posiciones de las partículas será de menor magnitud [38].

Con respecto a la diversidad dentro del enjambre la relación es inversa. Dado que la mejor posición encontrada se propaga más rápido en vecindades grandes (e.g. modelo global), mayor cantidad de partículas serán atraídas hacia ella, disminuyendo la diversidad del enjambre rápidamente de una iteración a otra. Con vecindades pequeñas (e.g. modelo local), sin embargo, las

partículas serán atraídas por las diferentes mejores posiciones correspondientes a cada una de las vecindades, de modo que el enjambre puede realizar una búsqueda en paralelo. Esto implica una estrategia de búsqueda más exhaustiva [36] y una convergencia más lenta pero con mayor diversidad [38].

De estas relaciones se puede establecer que el modelo global converge más rápido, pero es más sensible al estancamiento en óptimos locales. Por su parte, el modelo local tiene más posibilidades de escapar de la región de un óptimo local pero al costo de una menor velocidad de convergencia [38].

En una comparación realizada entre estos dos modelos [37], sobre un conjunto variado de funciones de prueba conocidas, se muestra que en iteraciones tempranas el rendimiento del modelo global supera al del modelo local para mayor cantidad de funciones, pero que luego de pasado cierto número de iteraciones, los resultados del modelo local superan a los del modelo global, especialmente en funciones multimodales.

En general, se plantea que la selección de la topología a utilizar es dependiente del problema a resolver y del número de iteraciones que se puedan emplear para optimizar. En problemas más simples como los unimodales, la utilización del modelo global puede ser conveniente, pues no existe el riesgo de una convergencia prematura en óptimos locales. Sin embargo, en problemas más complejos como los multimodales o multiobjetivos, el beneficio puede ser mayor con el empleo del modelo local y mayor número de iteraciones que permita garantizar una mayor exploración del espacio de búsqueda.

Teniendo en cuenta que no existe una topología idónea para todo tipo de problemas en [30] se sugiere la utilización de estos dos modelos de vecindades en caso de que se desee desarrollar una investigación rigurosa.

1.4. Versión estándar de PSO

Desde su introducción como técnica de optimización global en 1995, muchas de las investigaciones con PSO se han concentrado en su perfeccionamiento, realizando pequeñas modificaciones o remodelaciones completas de la idea original, así como en su utilización como punto de comparación para el perfeccionamiento de otras metaheurísticas (e.g. Algoritmos Genéticos, Evolución Diferencial, etc.). Hasta hace unos años no existía un acuerdo general entre los investigadores de lo que constituía la versión estándar de PSO. Bajo el mismo nombre de PSO, las implementaciones de esta técnica variaban considerablemente de una publicación a otra y muchas de las

variantes utilizadas para realizar comparaciones con otras técnicas de optimización, no tomaban en cuenta la mayoría de las modificaciones propuestas desde su creación.

Muchas de estas modificaciones representaban mejoras significativas al algoritmo y, aunque habían sido ampliamente conocidas y usadas, no se habían adoptado universalmente. El objetivo de D. Bratton y J. Kennedy en [30], fue precisamente, examinar las principales mejoras realizadas al algoritmo desde su creación y definir el nuevo estándar de PSO, el cual fue enunciado con las siguientes características:

- Uso del modelo local (lbest) como estructura de vecindad.
- Uso del factor de constricción en la ecuación para actualizar las velocidades (1.12) y la ecuación (1.6) para actualizar las posiciones.
- Enjambre compuesto por 50 partículas.
- Inicialización no uniforme del enjambre.
- Condiciones de frontera tales que las posiciones no factibles no son evaluadas.

Al decir de sus creadores, *"La intención de esta definición no es desalentar la exploración y alteraciones en el algoritmo PSO, sino más bien dar a los investigadores una base común para trabajar. Esta base común proporcionará un medio de comparación para futuras mejoras y evitará que esfuerzos innecesarios sean dedicados a "reinventar la rueda" (...)"* y señalan: *"Esta nueva definición de PSO no debe ser considerada como la "mejor opción posible" para todos los conjuntos de problemas. Existe un gran número de variaciones e hibridaciones del algoritmo original que potencialmente ofrecen un mejor rendimiento sobre determinados problemas. Lo que se ofrece aquí es a la vez un estándar para la comparación de algoritmos y un estándar para representar el PSO moderno en la comunidad de la optimización global."* [30]

Dado que el objetivo de esta investigación consiste en crear nuevas estrategias para mejorar el balance entre exploración y explotación de PSO y, como consecuencia, su rendimiento, entonces se tomará como punto de comparación esta nueva definición del estándar de PSO con implementaciones de los modelos de vecindades, local y global, como se sugiere en [30].

1.5. Estrategias propuestas para mejorar el rendimiento de PSO

En esta sección se presentarán algunas de las estrategias propuestas para perfeccionar el rendimiento de PSO en problemas complejos como los de grandes dimensiones y multimodales. Estas serán analizadas mediante la descripción de diferentes algoritmos basados en la versión estándar de PSO, al cual le han incorporado técnicas para mejorar el balance entre la exploración y explotación del espacio de búsqueda.

1.5.1. Estrategia de utilización eficiente de la población en Optimización de Enjambre de Partículas (EPUS-PSO)

El EPUS-PSO [19] es una variante de PSO estándar con topología de estrella. Este algoritmo resulta interesante y novedoso pues introduce un gestor de población para ajustar el tamaño del enjambre de acuerdo al estado de la búsqueda y, además, estrategias para el intercambio de soluciones y rangos de búsqueda, lo cual permite a las mejores partículas intercambiar sus experiencias unas con otras. El objetivo principal de cada una de estas estrategias es incrementar la exploración del espacio de soluciones y evitar una convergencia prematura del algoritmo, lo cual es más probable en el modelo global de PSO.

Gestor de población

El tamaño del enjambre en el EPUS-PSO es variable. Si las partículas no pueden encontrar una mejor solución para actualizar *gbest*, es probable que hayan sido atraídas por un óptimo local o necesiten ser guiadas hacia una región con más potencial. Asumiendo que la información (experiencia) de las partículas actuales es muy pobre para manejar la búsqueda eficazmente, en este caso el gestor de población añade nuevas partículas al enjambre para garantizar una mayor exploración en otras regiones. Por otro lado, si el *gbest* es actualizado durante varias iteraciones consecutivas una causa pudiera ser que las partículas actuales sean demasiadas para realizar la búsqueda y en este caso las de peor evaluación o redundantes son eliminadas.

Intercambio de soluciones

La estrategia de intercambio de soluciones propone una variación en la ecuación empleada para actualizar la velocidad de las partículas:

$$v_{i,j}^{t+1} = \begin{cases} wv_{i,j}^{t} + c_1 r_{1_{i,j}}^{t}[pbest_{i,j}^{t} - x_{i,j}^{t}] + c_2 r_{2_{i,j}}^{t}[gbest_{i,j}^{t} - x_{i,j}^{t}] & \text{si} \quad rand \geq P_{S_i} \\ wv_{i,j}^{t} + c_1 r_{1_{i,j}}^{t}[pbest_{i,j}^{t} - x_{i,j}^{t}] + c_2 r_{2_{i,j}}^{t}[pbest_{r,j}^{t} - x_{i,j}^{t}] & \text{si} \quad rand < P_{S_i} \end{cases} \tag{1.16}$$

Donde P_{S_i} es la probabilidad de intercambio de soluciones de la partícula i y decidirá si la partícula se mantendrá atraída por el *gbest* o lo será por el *pbest* de cualquier otra partícula ($pbest_r$) sin incluirse a sí misma. De esta forma, cada partícula tendrá la oportunidad de beneficiarse de la experiencia personal de otras partículas y disminuirá la velocidad de convergencia del algoritmo.

Intercambio de rangos de búsqueda (SRS)

Con el objetivo explícito de perfeccionar la exploración en el área para producir más soluciones potenciales, encontrar espacios de soluciones no explorados y así prevenir el estancamiento en óptimos locales, se introduce la estrategia de intercambio de rangos de búsqueda (SRS). La estrategia SRS se asemeja al operador de mutación en los algoritmos evolutivos [40] y puede activarse bajo un índice predefinido (SRS rate) que puede variar linealmente desde 0 hasta 0,2 durante el transcurso de la corrida del algoritmo (similar al mecanismo de ajuste de la temperatura en Recocido Simulado). En esencia, las posiciones de las partículas son perturbadas para comenzar a buscar en otra región del espacio de búsqueda.

SRS se clasifica en dos versiones: local y global. En la versión local, las nuevas posiciones de las partículas son restringidas a los límites de las mejores soluciones encontradas en el pasado ($[pbest_{min}, pbest_{max}]$), mientras que en la versión global se restringen a los límites de búsqueda iniciales ($[X_{min}, X_{max}]$). En tanto la versión local enfatiza la explotación de una región ya examinada la versión global favorece la exploración de nuevas regiones. Esta estrategia también se propone para el intercambio de rangos de búsqueda de dimensiones.

1.5.2. Optimización de Enjambre de Partículas con Múltiples Enjambres Dinámicos (DMS-PSO)

El DMS-PSO [20] es una variante del modelo local de PSO en la que se introduce una topología de vecindades dinámicas y aleatorias combinada con el concepto de subpoblaciones, de manera similar al empleado en los algoritmos evolutivos como estrategia para reducir la velocidad de convergencia y aumentar la diversidad de soluciones en la población.

Sub-enjambres y estrategia de reagrupamiento

La nueva topología se apoya en la división del enjambre de partículas en varios sub-enjambres, con la intención de que cada uno explore una región diferente del espacio de búsqueda. Cada sub-enjambre se compone de un número pequeño de partículas con el objetivo de diversificar la búsqueda. Con cierta frecuencia, para reducir la probabilidad de una convergencia prematura, se produce un período de reagrupamiento donde las partículas son reubicadas aleatoriamente favoreciendo el intercambio de información entre ellas y se reinicia la búsqueda con la nueva configuración de vecindades.

Combinación con búsqueda local

La estrategia de realizar migraciones entre los sub-enjambres produce un incremento en la diversidad y exploración del espacio de búsqueda pero también disminuye la velocidad de convergencia del enjambre, lo cual no es deseable si se ha llegado a la región del óptimo global. Para balancear este exceso de exploración DMS-PSO utiliza un algoritmo de búsqueda local para refinar las mejores soluciones. En particular, aplica el método BFGS Quasi-Newton sobre las posiciones *gbest* de los mejores sub-enjambres cada cierto número de iteraciones y sobre la mejor solución encontrada al finalizar el algoritmo.

El PSO es un algoritmo de búsqueda global y por tanto no hay seguridad de que la solución final que provee sea la mejor solución de su vecindad de ahí que la utilización de un método exacto para refinar la mejor solución encontrada sea una estrategia ampliamente utilizada. Como esta variante de PSO existen otras [22] que han experimentado con el empleo de métodos exactos para refinar la búsqueda global.

Fase de exploración y fase de convergencia

DMS-PSO propone además, dividir el procedimiento de ejecución del algoritmo en una fase de exploración y una de convergencia. En la fase de exploración las partículas examinan diferentes regiones del espacio de búsqueda simultáneamente mediante la subdivisión del enjambre, mientras que en la fase de convergencia – en la que se intensifica la búsqueda en la mejor región del espacio encontrada durante la fase de exploración – todas las partículas pertenecen a un mismo enjambre. Para pasar de una fase a otra se verifica el cumplimiento de algún criterio de convergencia, por ejemplo que la distancia máxima entre las mejores posiciones de las partículas (*pbest*) sea menor que un valor predefinido.

1.5.3. Olas de Enjambres de Partículas (WoSP)

Con la simulación de fuerzas gravitacionales en incremento, el algoritmo WoSP [21] añade a PSO la capacidad de explorar varios óptimos durante el proceso de búsqueda. Estas fuerzas causan entre dos partículas cercanas un incremento en sus velocidades, una en dirección a la otra, de modo que, las partículas son lanzadas en direcciones opuestas en el próximo intervalo de tiempo. De esta forma, cuando las partículas comienzan a converger alrededor de un óptimo estas se dispersan formando olas de partículas parcialmente independientes favoreciendo la exploración de otras regiones en busca de nuevos óptimos.

Fuerzas de atracción y efecto de repulsión

Para redirigir el movimiento de dos partículas que están convergiendo sobre una misma región del espacio (idealmente alrededor de un óptimo local), WoSP introduce una atracción semejante a la fuerza de gravedad, desde la partícula i hacia la j, cuya magnitud es inversamente proporcional a alguna potencia p de la distancia entre ellas. La fuerza de atracción (SRF) producirá un efecto de repulsión (Figura 1.3) entre las partículas debido al aumento de sus velocidades.

El cálculo de una componente del vector velocidad de una partícula en dirección a la otra está dado por la ecuación:

$$v_{ij} = \frac{K}{d_{ij}^{p}} \tag{1.17}$$

Donde K es una constante y d_{ij} es la distancia entre las partículas i y j.

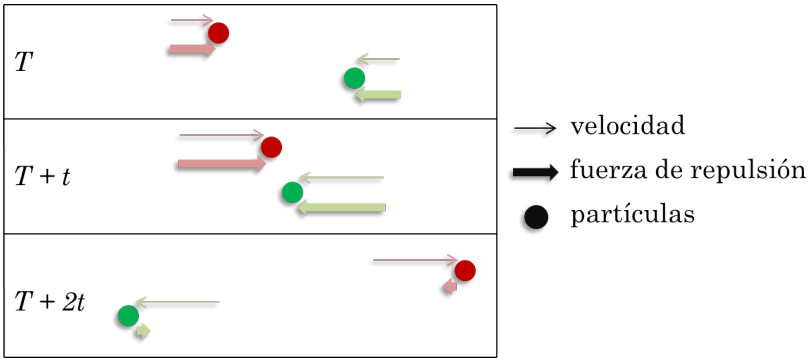


Figura 1.3: Efecto de las fuerzas de repulsión en WoSP

Organización de partículas en olas

El efecto de repulsión para alejar a las partículas de regiones de óptimos locales debe ser combinada con otras estrategias que permitan evitar que regresen a la región de la cual han sido expulsadas, lo cual sería de esperar debido a que es la mejor región que han encontrado hasta el momento y todavía tienen conocimiento de ello. En la solución que propone WoSP las partículas son organizadas en olas, donde cada partícula responderá a las mejores soluciones encontradas por los miembros de su ola. En este sentido, WoSP puede verse como otra variante de PSO con múltiples enjambres que se van creando dinámicamente.

El número de olas y la cantidad de partículas que las componen varía a medida que las partículas van convergiendo y por tanto van siendo expelidas. Inicialmente todas las partículas pertenecen a una misma ola (la número 0). Cada vez que una partícula es expelida se promueve a la ola siguiente, creando una nueva si se encuentra en la de mayor numeración, y se almacena su posición actual en su lista de puntos de promoción. El comportamiento de cada partícula en todo momento dependerá de su distancia hacia el más cercano de sus puntos de promoción. Si tal distancia es mayor que un umbral predefinido (rango de búsqueda), entonces se le confiere a la partícula la capacidad de informar su valor y posición a los demás miembros de su ola. Solo a las partículas con esta capacidad se les permite actualizar su velocidad de la manera estándar, pero respecto a la mejor solución encontrada por los miembros de su ola. Las restantes partículas, como se encuentran muy cercanas a algunos de sus puntos de promoción, los cuales representan posiblemente regiones de óptimos locales, actualizan su velocidad teniendo en cuenta su velocidad actual y el punto de promoción más cercano.

Similar a la lista tabu en la metaheurística Búsqueda Tabú [4, 5, 6], la lista de puntos de promoción en WoSP permite favorecer la exploración de nuevas regiones prohibiendo las ya examinadas. También la existencia de múltiples olas de partículas, cada una respondiendo a diferentes posiciones *gbest*, al igual que el modelo local de PSO, favorece la exploración simultánea de diferentes regiones del espacio de búsqueda reduciendo la probabilidad de una convergencia prematura.

1.5.4. Locust Swarms

Locust Swarms [22] se basa en el algoritmo WoSP y su diseño se apoya en la hipótesis de que la trayectoria ideal de la búsqueda para problemas multimodales debe mostrar un comportamiento no convergente y no aleatorio.

Los intervalos de tiempo y la dirección de las partículas durante el proceso de dispersión en WoSP son seleccionados con un alto grado de aleatoriedad, desaprovechando un poco la información acumulada durante la búsqueda. En Locust Swarms, sin embargo, las olas de partículas son lanzadas, cada cierto número predefinido de iteraciones, hacia regiones específicas del espacio de soluciones determinadas teniendo en cuenta la información acumulada durante el proceso de búsqueda.

Estrategia "devorar y continuar"

La estrategia de Locust Swarm para explorar múltiples óptimos es *devorar y continuar*. Después de examinar (explotar) rigurosamente una región del espacio de soluciones, hasta converger en un óptimo local o una solución cercana a uno de ellos, el algoritmo usa partículas de exploración, *exploradores* – creadas a partir de la mejor solución previamente encontrada – como punto de partida para descubrir otras soluciones de calidad en regiones sin explorar.

Locust Swarms puede describirse como un algoritmo con reinicios con dos características fundamentales: (1) la generación del enjambre al reiniciar la búsqueda (excepto del inicial) no es completamente aleatoria sino que depende de la mejor solución encontrada en la fase de exploración recién finalizada y (2) en cada fase de exploración se usa un PSO estándar con topología de estrella pero donde las partículas solo son atraídas por la mejor posición encontrada por todo el enjambre (*gbest*). El argumento del autor para no aprovechar la experiencia personal de cada partícula es que con la utilización de varios sub-enjambres se logra equilibrar el carácter

de explotación que adquiere el algoritmo. En el algoritmo 2 se describe de manera general este procedimiento.

Algoritmo 2 Locust Swarms

1. Generar S partículas aleatoriamente.
2. Seleccionar las L mejores partículas para aplicar PSO.
3. Asignar velocidades aleatorias a cada partícula.
4. Aplicar PSO durante k iteraciones.
5. **for** $n = 2$ to N **do**
6. Generar S partículas aleatorias alrededor del último óptimo encontrado.
7. Seleccionar las L mejores partículas (exploradores) para aplicar PSO.
8. Asignar velocidades de expulsión a cada partícula.
9. Aplicar PSO durante k iteraciones.
10. **end for**
11. Retornar mejor solución encontrada

Generación de exploradores

La diferencia más significativa entre Locust Swarms y un algoritmo con reinicios tradicional es el uso de la experiencia acumulada en la siguiente fase de exploración. Esto se realiza mediante la generación de exploradores alrededor de la mejor solución encontrada hasta el momento (*gbest*) como partículas iniciales para recomenzar el proceso de búsqueda.

La generación de cada partícula de exploración se realiza mediante la ecuación (1.18), donde cada dimensión $j = 1 \dots D$ de su vector posición constituye una variación, de tamaño *delta*, de la dimensión j-ésima de la posición *gbest* anterior.

$$x_{i,j}^{0} = gbest_{j}^{t} + delta \tag{1.18}$$

$$delta = \pm range * gap + abs(randn() * spacing) \tag{1.19}$$

Variando el tamaño de este *delta* (1.19)[2], se puede intensificar la búsqueda alrededor de *gbest* (con valores de *gap* y *spacing* pequeños) o explorar en otras direcciones creando soluciones más diversas (con valores de *gap* y *spacing* más grandes). Otra forma de garantizar la exploración de otras áreas es mediante la asignación a los exploradores de velocidades iniciales en direcciones opuestas al *gbest* para expulsarlos fuera de la región recién examinada.

[2] *randn* es una función de MATLAB que genera números aleatorios con distribución normal

Locust Swarms hace énfasis en lograr una buena exploración del espacio de soluciones más que en garantizar la convergencia del algoritmo, de ahí que el paso de una fase de exploración a otra no dependa de algún criterio de convergencia sino de haber llegado a un número de iteraciones predefinido. No obstante, para mejorar los resultados del algoritmo y garantizar que la búsqueda reinicie teniendo en cuenta la información de una solución de más calidad (óptimo local), luego de cada fase de exploración y antes de generar los exploradores, la solución *gbest* puede ser refinada mediante la utilización de un algoritmo de búsqueda local[3], similar a como lo realiza el algoritmo DMS-PSO (Sección 1.5.2).

Reducción de la Dimensión (dim_r)

La estrategia de reducir la dimensión durante la creación de los exploradores es otra manera en Locust Swarms de balancear la exploración y explotación del espacio de búsqueda.

La interpretación de este nuevo parámetro – denominado dim_r – es la siguiente: dim_r es igual a la cantidad de términos/dimensiones que son modificadas por *delta* durante la creación de los exploradores. Por ejemplo, en un problema de dimensión 30 si $dim_r = 2$ entonces, al crear un explorador, solo 2 dimensiones de su posición, seleccionadas aleatoriamente, serán modificadas de acuerdo a (1.18) y las restantes 28 dimensiones permanecerán con el valor correspondiente al de la solución *gbest* encontrada en la fase de exploración anterior.

Entre este parámetro y el operador de mutación de los algoritmos evolutivos hay cierta similitud en cuanto a los efectos que producen. Similar al operador de mutación, mediante el cual se puede controlar la diversidad en la población variando la probabilidad de mutación (p_m), un valor grande de dim_r implica la creación de exploradores con más diferencias respecto al *gbest*, y lo contrario si se utiliza un valor pequeño. De esta forma, decrementando el valor de dim_r se intensifica la búsqueda alrededor de la solución *gbest*, mientras que su incremento favorece la exploración de otras soluciones.

[3]El autor en [22] utiliza la función $fmincon$ de MATLAB. Es un método basado en gradientes diseñado para minimizar funciones de varias variables con restricciones. Las funciones y las restricciones pueden ser funciones no lineales.

1.5.5. Discusión

Aunque existen otras técnicas [41, 42] con propuestas más o menos diferentes de las que se han descrito aquí, a todas las une un mismo propósito: mejorar el balance entre la exploración y explotación en PSO, y como consecuencia, hacer de este un algoritmo más potente y robusto.

El objetivo de este trabajo no es realizar comparaciones entre las técnicas existentes, sino proponer otras estrategias que permitan mejorar los resultados del algoritmo, así como profundizar en el conocimiento que se tiene sobre su funcionamiento.

Las técnicas que se proponen pueden servir a otros investigadores en el área para resolver problemas específicos o como motivación para la creación de otras nuevas, tal y como sucedió con los resultados del algoritmo Locust Swarms [22, 43, 44] que sirvieron para la experimentación con una nueva variante de PSO que será presentada en el próximo capítulo. Es por ello que se considera necesario abordar en detalle algunos de estos resultados.

Resultados de Locust Swarms

En [22] se presentan las comparaciones realizadas entre Locust Swarms y los algoritmos Búsqueda Aleatoria [45] y Algoritmos Meméticos [46, 47]. Tomando como función de prueba la función Schwefel[4] en un espacio de 30 dimensiones y sin reducir la dimensión durante la creación de los exploradores ($dim_r = 30$), los resultados de Locust Swarms, comparados respecto al promedio de los valores de la función y la desviación estándar, fueron superiores, aunque la cantidad de soluciones óptimas encontrada por los tres algoritmos fue igual a 0.

Las pruebas realizadas reduciendo la dimensión ($dim_r = 1, 2, 3, 5, 10$) revelaron, sin embargo, resultados más interesantes. Valores altos de dim_r implicaban una exploración más aleatoria con menor explotación de la información acumulada durante la búsqueda y, a medida que se aumentaba la reducción de la dimensión, la cantidad de soluciones óptimas encontradas era mayor haciendo notar la capacidad del algoritmo para explotar la separabilidad[5] de la función de prueba.

[4] Función separable con múltiples óptimos (alrededor de 2^D óptimos donde D es el tamaño de la dimensión del espacio).[48]

[5] Generalmente las funciones de prueba de grandes dimensiones se crean mediante la repetición (e.g. suma, multiplicación) de varias funciones de una dimensión. El resultado es una función *separable* cuyo óptimo global se puede obtener resolviendo por separado las funciones de una dimensión. En este sentido, las funciones no separables se consideran más difíciles de resolver y, en consecuencia, muchas funciones de prueba son creadas no

Por otro lado, al reducir la separabilidad[6] de la función, los mejores resultados (mayor cantidad de soluciones óptimas distintas) se obtuvieron al lograr, con $dim_r = 5$, un mejor balance entre la no convergencia y no aleatoriedad en la búsqueda. Con mayor reducción ($dim_r = 1, 2, 3$) esta se hacía más convergente resultando en menor cantidad de soluciones óptimas diferentes al igual que con menor reducción o ninguna ($dim_r = 10, 30$), pero debido a que la búsqueda se hacía más aleatoria. Estos resultados señalaron a dim_r y su utilización en la generación de los exploradores como una de las componentes más importantes del algoritmo y con más influencia en lograr un buen balance entre la exploración y explotación del espacio de búsqueda.

En [43] se hace un análisis de Locust Swarms al resolver problemas de grandes dimensiones y se compara con otros algoritmos, entre ellos EPUS-PSO y DMS-PSO (secciones 1.5.1 y 1.5.2), utilizando las funciones de prueba de la competencia LSGO[7]. Los resultados de Locust Swarms en estas funciones fueron muy buenos y, en particular, mucho mejores a los de EPUS-PSO y DMS-PSO. A diferencia de estos, de una manera u otra los demás algoritmos presentados en la competencia, junto a Locust Swarms, explotaban la separabilidad de una función, lo cual confirmó a la estrategia de la reducción de la dimensión como el factor más influyente en el rendimiento de Locust Swarms y posiblemente en el de otros algoritmos.

La publicación [44] es el resultado de una investigación desarrollada en conjunto con el autor del algoritmo[8] en la que se analizan los efectos de la reducción de la dimensión en el rendimiento de PSO, tomando como caso de estudio a Locust Swarms y 24 funciones de prueba de *caja-negra* (Black-Box Optimization Benchmarking Problems [48]) aunque se prestó especial atención al conjunto de funciones separables y a las funciones multimodales con estructura global adecuada.

Los resultados obtenidos revelaron la superioridad de Locust Swarms sobre el estándar de PSO incluso en las funciones no separables y para grandes dimensiones. Aunque de manera general, el uso de la reducción de la dimensión favorecía a Locust Swarms, en las funciones separables los mejores resultados se obtuvieron para valores pequeños de dim_r, mostrando la efectividad de esta estrategia para resolver funciones separables. Por otro lado,

separables [48].

[6] La técnica establecida para reducir la separabilidad de una función es mediante la aplicación de rotaciones en el espacio a la función [48]. De esta forma, se pueden generar funciones no separables a partir de funciones separables.

[7] *Optimización Global de Problemas de Grandes Dimensiones* [49], Congreso de Computación Evolutiva (IEEE CEC).

[8] Stephen Chen, School of Information Technology, York University, Toronto, Canada.

en la funciones no separables y multimodales los resultados para $dim_r = 1$ fueron mucho peores que para $dim_r = D$, indicando que limitar el proceso de búsqueda en estas funciones a una sola dimensión no trae beneficios y que al menos con $dim_r = D$ puede garantizarse una mayor exploración del espacio de búsqueda. Para estas funciones los mejores resultados se obtuvieron con valores intermedios de dim_r, confirmando los resultados obtenidos en [22].

También se realizaron pruebas para medir el efecto de la *maldición de la dimensionalidad*[9] en el algoritmo. Para ello se obtuvieron los resultados para diferentes escalas ($D = 20, 50, 100, 200$) y se compararon con los obtenidos por una implementación estándar de PSO. Como conclusión se llegó a que Locust Swarms también es afectado por este fenómeno pero en menor magnitud que el algoritmo utilizado como punto de comparación, el cual no posee estrategias explícitas para explotar la separabilidad de una función.

Reducción de la Dimensión en PSO

Según los resultados anteriores, la reducción de la dimensión es un componente fundamental en el diseño de Locust Swarms que le permite superar el rendimiento de PSO estándar, para diferentes valores de dim_r dependiendo de las características de la función objetivo a optimizar. Considerando que la reducción de la dimensión no forma parte del proceso de búsqueda del PSO utilizado en Locust Swarms, surge la interrogante de qué beneficios podría aportar esta estrategia al PSO en términos de mejorar su balance entre exploración y explotación.

En este contexto, y ante la ausencia de referencias en la literatura consultada, respecto a la inclusión de esta estrategia como parte del proceso de búsqueda de PSO, es que surge la idea de diseñar una variante de este algoritmo con tales características. En el siguiente capítulo se describe una propuesta de PSO con Reducción de la Dimensión.

[9] El término *maldición de la dimensionalidad* (curse of dimensionality, en inglés) se utiliza para refererirse al fenómeno que afecta a la mayoría de los algoritmos de optimización cuando su rendimiento se deteriora rápidamente a medida que se incrementa la dimensión del espacio [49].

Capítulo 2

Optimización de Enjambre de Partículas con Reducción de la Dimensión

Del estudio del algoritmo Locust Swarms [22] y el análisis de sus resultados [43, 44] surgió la idea de diseñar una nueva variante de la versión estándar de PSO en la que se incluyera la estrategia de la reducción de la dimensión dentro del proceso de búsqueda del algoritmo. La motivación especial para realizar esta investigación, además de que no existían referencias en la literatura de un algoritmo con estas características, fue determinar cuánto podía beneficiar esta estrategia al proceso de exploración de un PSO estándar, tanto global como local, y realizar comparaciones para obtener más información respecto al funcionamiento de cada uno. El resultado de esta investigación es el algoritmo ManhattanPSO, cuya descripción se presenta a continuación.

2.1. Algoritmo ManhattanPSO

El algoritmo ManhattanPSO (MPSO) que se describe en esta sección se basa en el concepto de Distancia Manhattan para modificar la trayectoria de las partículas y reducir el número de dimensiones analizadas durante el proceso de búsqueda de PSO.

Para ilustrar la trayectoria que una partícula haría siguiendo el camino de las distancias Manhattan en un espacio de 2 dimensiones y reduciendo la búsqueda a una dimensión, supóngase que se tiene una partícula en la posi-

ción $(1,0)$ (Figura 2.1) y que sus próximos k movimientos están dirigidos a alcanzar la posición $(0,1)$. En un PSO estándar la trayectoria de la partícula sería en línea recta (la línea sólida de color negro, siguiendo el camino de la distancia Euclidiana), mientras que siguiendo el camino de las distancias Manhattan (líneas discontinuas de color rojo) podría llegarse a la posición $(0,0)$ manteniendo fija la dimensión y y explorando la dimensión x o a la posición $(1,1)$ explorando la dimensión y y manteniendo fija la x.

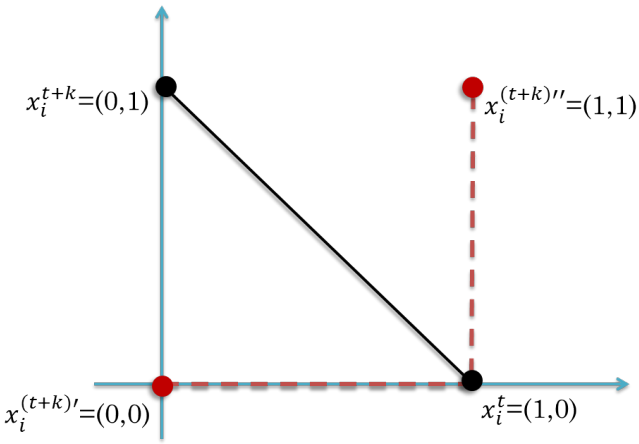


Figura 2.1: Trayectoria de una partícula siguiendo el camino de las distancias Manhattan

Las ecuaciones para actualizar la velocidad y posición de una partícula en MPSO son las mismas que las de un PSO estándar (ecuaciones (1.12) y (1.6)), sin embargo, el proceso de exploración se divide en períodos de k iteraciones. Durante las k iteraciones que dura un período las partículas serán enfocadas a buscar el óptimo para una subconjunto de las dimensiones. Para el siguiente período el conjunto de dimensiones objetivo cambiará y el proceso de exploración continuará, pero enfocándose en las dimensiones recién seleccionadas. Se espera que la ejecución del algoritmo durante un número suficiente de iteraciones, con una selección adecuada de las dimensiones en cada período, pueda garantizar la exploración de todas las dimensiones del espacio y finalmente encontrar la solución óptima para el problema multidimensional. La descripción por pasos de este procedimiento se muestra en el algoritmo 3.

La variación más importante que se hace al estándar de PSO tiene que ver con la evaluación de las partículas (pasos 8 y 9). Aunque la trayectoria de una partícula sigue siendo definida por su posición estándar x_i, la evaluación

Algoritmo 3 ManhattanPSO (MPSO)

1. Inicializar constantes y variables (D, dim_r, S, k, T, x_i^0, v_i^0, ...)
2. $pinitial_i \leftarrow x_i^0$
3. $DIM_r \leftarrow$ seleccionar dim_r dimensiones de D
4. **for** $t = 1$ to T **do**
5. **for** $i = 1$ to S **do**
6. Actualizar v_i^t usando ecuación (1.12)
7. Actualizar x_i^t usando ecuación (1.6)
8. Crear posición $pevaluate_i$ usando Algoritmo 4
9. Evaluar $pevaluate_i$ y actualizar $pbest_i$ usando ecuación (2.1)
10. **end for**
11. **for** $i = 1$ to S **do**
12. Actualizar $gbest_i$ usando ecuación (1.8)
13. **end for**
14. **if** $t \equiv 0$ (mód k) **then**
15. $pinitial_i \leftarrow pevaluate_i$
16. $DIM_r \leftarrow$ seleccionar dim_r dimensiones de D
17. **end if**
18. **end for**

de la función objetivo en la posición $pevaluate_i$ (posición de evaluación de la partícula i) permite que otras trayectorias sean las que influyan en el proceso de búsqueda (e.g. líneas discontinuas rojas en la figura 2.1).

El cálculo de la posición de evaluación de una partícula i se realiza tomando dim_r dimensiones objetivo de x_i y $(D - dim_r)$ dimensiones de $pinitial_i$ como se muestra en el algoritmo 4. En $pinitial_i$ se almacena la posición inicial de la partícula i antes de comenzar el nuevo período de exploración. De esta forma, durante k iteraciones, la exploración del espacio de búsqueda se verá restringida a una parte de las dimensiones fijada por la selección del conjunto de dimensiones objetivo DIM_r.

Algoritmo 4 Cálculo de la posición de evaluación de una partícula

Entrada: x_i, $pinitial_i$, D, DIM_r
Salida: $pevaluate_i$

1. **for** $d = 1$ to D **do**
2. **if** $d \in DIM_r$ **then**
3. $pevaluate_i^d = x_i^d$
4. **else**
5. $pevaluate_i^d = pinitial_i^d$
6. **end if**
7. **end for**

Esto se puede apreciar en la figura 2.2, que ilustra la trayectoria de una partícula en un espacio de 2 dimensiones durante 4 iteraciones. Los círculos

negros representan las posiciones x_i obtenidas en cada iteración, explorando las dos dimensiones del espacio, mientras que los círculos rojos, representan las posiciones de evaluación $pevaluate_i$ que han sido creadas tomando la primera dimensión ($DIM_r = \{1\}$) de la posición x_i y la segunda dimensión ($D - DIM_r = \{2\}$) de la posición inicial $pinitial_i$, o sea, explorando la primera dimensión.

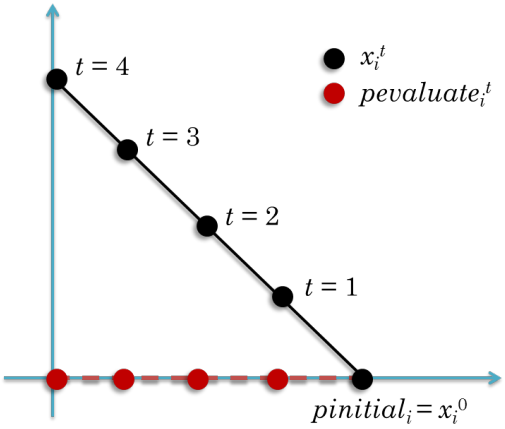


Figura 2.2: Posiciones de evaluación consideradas en MPSO

La actualización de las posiciones $pbest_i$ y $gbest_i$ (ecuaciones (2.1) y (1.8)) se realiza tras la evaluación de todas las partículas del enjambre, pero a diferencia de la versión estándar de PSO, en sus actualizaciones se tienen en cuenta las posiciones $pevaluate_i$ y no las posiciones x_i.

$$pbest_i = \begin{cases} pevaluate_i & \text{si} \quad f(pevaluate_i) < f(pbest_i) \\ pbest_i & \text{si} \quad f(pevaluate_i) \geq f(pbest_i) \end{cases} \tag{2.1}$$

Esta forma de actualizar el *pbest* y *gbest* de cada partícula, así como la actualización de *pinitial* con las posiciones *pevaluate*, permitirá usar la información acumulada sobre el conjunto de dimensiones recién explorado en los próximos períodos, aunque la exploración se lleve a cabo sobre un nuevo conjunto de dimensiones.

Otro aspecto que distingue a MPSO de una implementación estándar de PSO es la selección del conjunto DIM_r en cada período. A continuación se describirán tres variantes de este algoritmo, que difieren en el mecanismo utilizado para seleccionar este conjunto.

2.1.1. MPSO con mecanismo de Selección Aleatoria y Reemplazo (SACR-MPSO)

Esta variante de MPSO es una de las más simples e intuitivas. Se denomina MPSO con mecanismo de Selección Aleatoria y Reemplazo (SACR-MPSO), pues el proceso de escoger las dimensiones es aleatorio y no se tienen en cuenta las dimensiones que han sido exploradas hasta el momento. Si se denomina muestra al resultado de escoger aleatoriamente dim_r dimensiones de una población de D dimensiones, se puede asociar este mecanismo de selección con el muestreo con reemplazo que se define en probabilidades.

Por ejemplo, si $D = 8$ y $dim_r = 3$, la selección de DIM_r se realiza sobre el conjunto $\{1,2,3,4,5,6,7,8\}$, de modo que un posible resultado para el primer período de exploración pudiera ser $DIM_r = \{1,2,6\}$. Para el segundo período, el proceso de selección de las dimensiones es aleatorio y se realiza sobre este mismo conjunto, considerando las dimensiones que fueron seleccionadas en el período anterior. De esta forma, el resultado bien podría repetir una dimensión ($DIM_r = \{4,2,8\}$), ser completamente diferente ($DIM_r = \{3,5,7\}$) o, incluso, coincidir con la selección anterior ($DIM_r = \{1,2,6\}$).

La descripción de esta variante en seudocódigo es similar al algoritmo MPSO descrito anteriormente (Algoritmo 3). Solo habría que especificar, en los pasos 3 y 16, que la selección de las dimensiones es aleatoria sobre todo el conjunto de dimensiones posibles a considerar, o sea D.

2.1.2. MPSO con mecanismo de Selección Aleatoria sin Reemplazo (SASR-MPSO)

La idea general de esta variante es muy similar a la anterior, pero se asocia con el muestreo sin reemplazo, de ahí su nombre. Esto significa que en SASR-MPSO (Algoritmo 5), para seleccionar un nuevo conjunto de dimensiones a explorar, no se toman en cuenta las dimensiones seleccionadas anteriormente mientras existan dimensiones que no han sido exploradas.

La selección del nuevo conjunto de dimensiones se mantiene aleatoria pero sobre un conjunto de dimensiones temporal R que almacena las dimensiones que restan por explorar. Una vez completado el número de períodos necesario para que todas las dimensiones sean exploradas el conjunto temporal se hace igual a su valor inicial $R = D$ y todas las dimensiones vuelven a ser consideradas.

Por ejemplo, si $D = 10$ y $dim_r = 4$, una posible selección del conjunto DIM_r para el primer período de exploración puede dar como resultado $DIM_r = \{1,2,6,10\}$. Esto significa que para los próximos períodos ninguna

Algoritmo 5 MPSO con Selección Aleatoria sin Reemplazo (SASR-MPSO)

1. Inicializar constantes y variables (D, dim_r, S, k, T, x_i^0, v_i^0, ...)
2. $pinitial_i \leftarrow x_i^0$
3. $R \leftarrow D$
4. $DIM_r \leftarrow$ combinación aleatoria de dim_r dimensiones en R
5. **for** $t = 1$ to T **do**
6. **for** $i = 1$ to S **do**
7. Actualizar v_i^t usando ecuación (1.12)
8. Actualizar x_i^t usando ecuación (1.6)
9. Crear posición $pevaluate_i$ usando Algoritmo 4
10. Evaluar $pevaluate_i$ y actualizar $pbest_i$ usando ecuación (2.1)
11. **end for**
12. **for** $i = 1$ to S **do**
13. Actualizar $gbest_i$ usando ecuación (1.8)
14. **end for**
15. **if** $t \equiv 0$ (mód k) **then**
16. $pinitial_i \leftarrow pevaluate_i$
17. $R \leftarrow R - DIM_r$
18. **if** $|R| \geq dim_r$ **then**
19. $DIM_r \leftarrow$ combinación aleatoria de dim_r dimensiones en R
20. **else**
21. $d = dim_r - |R|$
22. $DIM_r \leftarrow R \cup \{$combinación aleatoria de d dimensiones en $D - R\}$
23. $R \leftarrow D$
24. **end if**
25. **end if**
26. **end for**

de estas dimensiones se tendrán en cuenta, o sea $R = \{3,4,5,7,8,9\}$. Si el conjunto seleccionado para el segundo período resulta ser $DIM_r = \{3,5,7,9\}$ entonces se actualizará $R = \{4,8\}$, de modo que para el próximo período se explorarán las dimensiones 4, 8 y otras dos seleccionadas aleatoriamente del conjunto actualizado $R = D - \{4,8\}$.

A diferencia de SACR-MPSO, esta variante asegura la exploración de todas las dimensiones al final de la ejecución del algoritmo y lo hace de una forma equilibrada, es decir, dedicando similar cantidad de períodos de exploración a cada dimensión. Es por ello que se espera obtener mejores resultados con esta variante.

2.1.3. MPSO con Búsqueda Exhaustiva (BE-MPSO)

Tanto SACR-MPSO como SASR-MPSO ambos tienen como característica que durante las k iteraciones que dura el período de búsqueda efectivo, la exploración del espacio de soluciones se realiza sobre el mismo subconjun-

to de dimensiones. A diferencia de ellos, el algoritmo MPSO con Búsqueda Exhaustiva (BE-MPSO) no posee un mecanismo para seleccionar el conjunto de dimensiones a considerar en cada período de exploración, pues en cada iteración realiza una búsqueda exhaustiva al tener en cuenta todas las posibles combinaciones de dim_r dimensiones sobre D.

Como se observa en su descripción (Algoritmo 6) la cantidad de posiciones de evaluación que son generadas y evaluadas por esta variante es mucho mayor (pasos 7 - 11). Por ejemplo, para $dim_r = 1$ y $D = 20$ se calcularían, en cada iteración por cada partícula, 20 posiciones de evaluación las cuales se evaluarían y compararían con $pbest_i$ actualizándola de acuerdo a la ecuación (2.1) en caso de encontrarse alguna posición con mejor evaluación.

Algoritmo 6 MPSO con Búsqueda Exhaustiva (BE-MPSO)

1. Inicializar constantes y variables (D, dim_r, S, k, T, x_i^0, v_i^0, ...)
2. $pinitial_i \leftarrow x_i^0$
3. **for** $t = 1$ to T **do**
4. **for** $i = 1$ to S **do**
5. Actualizar v_i^t usando ecuación (1.12)
6. Actualizar x_i^t usando ecuación (1.6)
7. **for all** combinación C de dim_r dimensiones en D **do**
8. $DIM_r \leftarrow C$
9. Crear posición $pevaluate_i$ usando Algoritmo 4
10. Evaluar $pevaluate_i$ y actualizar $pbest_i$ usando ecuación (2.1)
11. **end for**
12. **end for**
13. **for** $i = 1$ to S **do**
14. Actualizar $gbest_i$ usando ecuación (1.8)
15. **end for**
16. **if** $t \equiv 0$ (mód k) **then**
17. $pinitial_i \leftarrow pbest_i$
18. **end if**
19. **end for**

Retomando el ejemplo gráfico usado anteriormente para ilustrar las posiciones de evaluación, en la figura 2.3 se ha dibujado, con círculos verdes, otro conjunto de posiciones de evaluación correspondiente a la combinación creada tomando la segunda dimensión ($DIM_r = \{2\}$) de la posición x_i y la primera dimensión ($D - DIM_r = \{1\}$) de la posición $pinitial_i$.

Mientras SACR-MPSO y SASR-MPSO evalúan por cada partícula una sola posición, la roja o la verde, en dependencia de la combinación de dimensiones $c_k \in C$ que haya sido seleccionada antes de comenzar el período de k iteraciones, BE-MPSO evalúa las dos posiciones. Además de asegurar la exploración de todas las dimensiones al final de la ejecución del algoritmo,

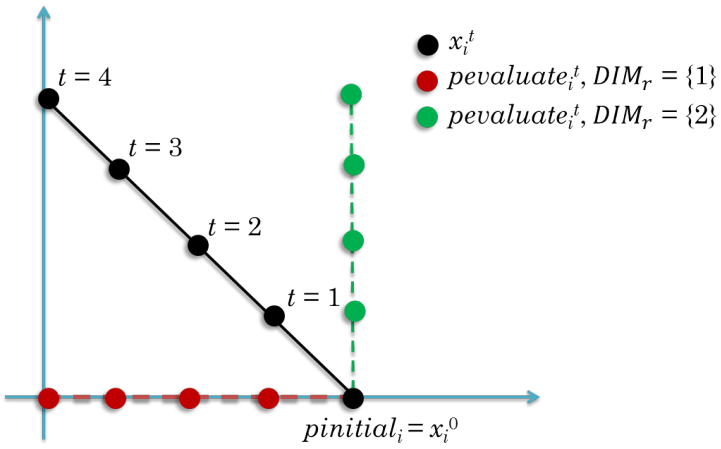


Figura 2.3: Posiciones de evaluación consideradas en BE-MPSO

la estrategia de búsqueda exhaustiva permite determinar la combinación de dimensiones que mejor evalúa la función objetivo. Su funcionamiento puede verse como un mecanismo para encontrar el conjunto de dimensiones que deben ser exploradas en cada momento.

2.2. Resultados Experimentales

En esta sección se describen los experimentos realizados para analizar los efectos que produce la Reducción de la Dimensión en PSO, tomando como caso de estudio los algoritmos descritos en este capítulo y la versión estándar de PSO.

Considerando que la estrategia propuesta está dirigida a mejorar el balance entre exploración y explotación en PSO, en los experimentos realizados se ha considerado el análisis y comparación de dos modelos diferentes de vecindades. Para cada uno de los algoritmos se han implementado dos versiones: una con topología de estrella (modelo global) y la otra con topología de anillo (modelo local con $n = 2$), para un total de 8 algoritmos, incluyendo las implementaciones de PSO estándar, que se utilizan como punto de comparación.

2.2.1. Entorno de desarrollo

Como entorno de desarrollo para realizar dichas implementaciones se usó Matlab, debido a las facilidades que brinda para el trabajo con matrices, graficar funciones y realizar pruebas estadísticas. Dado que las posiciones de las partículas son vectores D-dimensionales de valores reales, un enjambre se puede representar mediante una matriz de $S \times D$ donde S es el número de partículas y D el número de dimensiones. De esta forma, el proceso de actualización de las velocidades y posiciones de las partículas se puede realizar muy fácilmente utilizando las operaciones definidas para matrices.

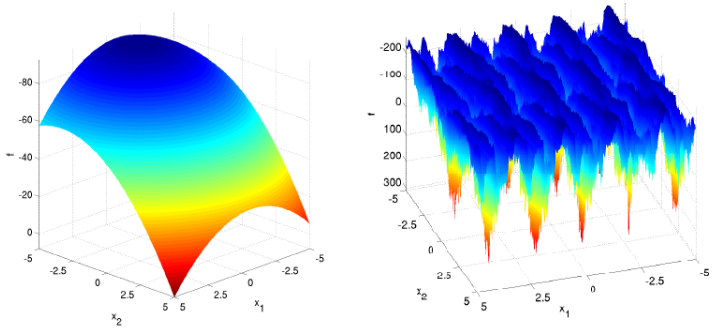
2.2.2. Funciones de prueba

Los experimentos se realizaron sobre una parte del conjunto de funciones de prueba de caja negra (Black-Box Optimization Benchmarking Problems (BBOB)) [48]. El objetivo de BBOB es proveer a los investigadores de una selección de funciones con algunas de las dificultades típicas de espacios continuos y que pueden verse con frecuencia en problemas reales. El análisis del comportamiento de un algoritmo frente a este tipo de funciones es muy utilizado en la actualidad[1] pues permite la autoevaluación del algoritmo, así como su comparación con otros en el área, sirviendo como mecanismo para su perfeccionamiento y el diseño de otros más eficientes. Las funciones BBOB se encuentran implementadas en Matlab, otro motivo por el cual se decidió utilizar este software matemático para implementar los algoritmos. En especial se seleccionaron de BBOB los conjuntos de funciones F1-F5 y F15-F19 debido a sus propiedades de separabilidad (s) y multimodalidad (m) (Tabla 2.1).

Tabla 2.1: Funciones de prueba utilizadas en los experimentos.

		Atributos	
Función	Nombre	s	m
1	Sphere	X	
2	Ellipsoidal	X	
3	Rastrigin	X	X
4	Büche-Rastrigin	X	X
5	Linear Slope	X	
15	Rastrigin rotada		X
16	Weierstrass		X
17	Schaffers		X
18	Schaffers moderadamente mal-condicionada		X
19	Griewank-Rosenbrock compuesta		X

[1] Por ejemplo, en las Conferencias de Computación Genética y Evolucionaria (GECCO) y los Congresos de Computación Evolucionaria (CEC)



(a) Función Sphere (b) Función Weierstrass

Figura 2.4: Ejemplo de funciones *Black-Box Optimization Benchmarking*

Como ejemplo de funciones de cada uno de estos conjuntos en la figura 2.4 se muestra una representación en tres dimensiones de las funciones Sphere y Weierstrass. La función Sphere es una de las funciones unimodales y separables más fáciles de resolver y la función Weierstrass, además de no ser separable, posee múltiples óptimos globales.

2.2.3. Diseño de los experimentos

Cada experimento consiste en realizar 50 ejecuciones de un mismo algoritmo sobre una misma función y configuración de parámetros: tamaño del enjambre $S = 50$, espacio de dimensiones $D = 20$ y número de evaluaciones de función $FE = 100000$. Estos son los parámetros comunes a todos los experimentos, los que son propios de uno en particular se describen más adelante.

Como resultado de cada experimento se calculan los promedios, para las 50 corridas, de los errores absolutos y relativos, de sus desviaciones estándares y de la iteración en la que se alcanza la mejor solución encontrada. También se obtienen de las 50 corridas la mejor y peor evaluación de la función objetivo. Se implementó, además, que cada algoritmo almacenara, cada 50 iteraciones, la mejor solución encontrada hasta el momento (*gbest*), con el objetivo de poder graficar la convergencia de cada uno.

Como se puede apreciar, es abundante la cantidad de información recogida en cada experimento. Esto permite hacer un análisis más profundo de la estrategia, pero al mismo tiempo, resulta difícil presentar todos los resultados de una manera adecuada, que permita entender los efectos más significativos que causa la misma al comportamiento de PSO. En este sen-

tido, para su análisis, se ha hecho una selección de los resultados obtenidos tratando de mostrar las observaciones que se consideran más relevantes. El resto de los resultados, aunque no se muestran en este informe por su extensión y detalle, se encuentra disponible para cualquier consulta. (Nota: esto es válido para el análisis que se realiza posteriormente de las estrategias que se proponen en el próximo capítulo).

2.2.4. Análisis de PSO estándar

Este primer experimento está dirigido a estudiar el comportamiento de la versión estándar de PSO con topología global (PSO_g) y la versión con topología local (PSO_l), para utilizarlas posteriormente como punto de comparación durante el análisis de las modificaciones propuestas al algoritmo.

Tabla 2.2: Resultados de PSO con topología global (PSO_g)

fn	Error abs.	Error rel.	Mejor	Peor	Des. Std.	Iteración
1	0.00E+00	0.00 %	0.00E+00	0.00E+00	7.22E-10	2.96E+02
2	1.91E+02	91.05 %	0.00E+00	4.11E+03	6.55E+02	5.73E+02
3	4.12E+01	8.91 %	9.95E+00	1.04E+02	1.93E+01	9.06E+02
4	4.84E+01	10.47 %	1.89E+01	9.55E+01	1.99E+01	1.00E+03
5	1.13E+01	123.07 %	0.00E+00	6.16E+01	1.42E+01	1.99E+01
15	6.26E+01	6.26 %	1.99E+01	1.45E+02	2.70E+01	1.54E+03
16	5.38E+00	7.54 %	1.19E+00	1.10E+01	2.12E+00	1.99E+03
17	1.39E+00	8.18 %	3.13E-01	3.38E+00	7.95E-01	1.76E+03
18	5.15E+00	30.39 %	6.46E-01	1.74E+01	3.59E+00	1.75E+03
19	3.37E+00	3.29 %	9.90E-01	5.11E+00	9.45E-01	1.30E+03

Tabla 2.3: Resultados de PSO con topología local (PSO_l)

fn	Error abs.	Error rel.	Mejor	Peor	Des. Std.	Iteración
1	0.00E+00	0.00 %	0.00E+00	0.00E+00	6.35E-10	6.46E+02
2	0.00E+00	0.00 %	0.00E+00	0.00E+00	1.04E-09	8.98E+02
3	3.14E+01	6.80 %	1.59E+01	5.43E+01	9.14E+00	1.97E+03
4	4.10E+01	8.87 %	1.99E+01	7.71E+01	1.02E+01	1.96E+03
5	0.00E+00	0.00 %	0.00E+00	0.00E+00	0.00E+00	4.11E+01
15	5.28E+01	5.28 %	2.98E+01	8.98E+01	1.45E+01	1.98E+03
16	6.32E+00	8.86 %	3.52E+00	9.46E+00	1.43E+00	1.97E+03
17	5.74E-01	3.39 %	7.00E-02	1.51E+00	2.93E-01	1.96E+03
18	3.09E+00	18.23 %	1.29E+00	6.73E+00	1.02E+00	1.96E+03
19	3.31E+00	3.23 %	2.37E+00	4.21E+00	5.08E-01	1.31E+03

Los resultados obtenidos por cada variante se muestran en las tablas 2.2 y 2.3 respectivamente. Como se puede observar en la comparación que

se realiza entre estos dos modelos (Tabla 2.4) el modelo local presenta, en promedio, un mejor rendimiento que el modelo global. La diferencia porcentual para cada función – calculada mediante la fórmula $(a-b)/a$, donde b representa al modelo local y a al modelo global – muestra la superioridad del modelo local en 8 funciones, la del modelo global solo en una y el empate en la primera función en la que los dos alcanzan el valor óptimo. De acuerdo a los test de prueba de hipótesis (t-test[2]) se confirma, además, que las diferencias entre los resultados obtenidos por ambos algoritmos son estadísticamente significativas para la mayoría de las funciones.

Tabla 2.4: PSO_g vs PSO_l

	PSO_g		PSO_l			
fn	Error abs.	Des. Std.	Error abs.	Des. Std.	dif-%	t-test
1	**0.00E+00**	7.22E-10	**0.00E+00**	6.35E-10	**0.00%**	**1.27%**
2	1.91E+02	6.55E+02	**0.00E+00**	1.04E-09	**100.00%**	**4.44%**
3	4.12E+01	1.93E+01	**3.14E+01**	9.14E+00	**23.74%**	**0.36%**
4	4.84E+01	1.99E+01	**4.10E+01**	1.02E+01	**15.28%**	**2.96%**
5	1.13E+01	1.42E+01	**0.00E+00**	0.00E+00	**100.00%**	**0.00%**
15	6.26E+01	2.70E+01	**5.28E+01**	1.45E+01	**15.62%**	**2.97%**
16	**5.38E+00**	2.12E+00	6.32E+00	1.43E+00	-17.57%	**0.60%**
17	1.39E+00	7.95E-01	**5.74E-01**	2.93E-01	**58.61%**	**0.00%**
18	5.15E+00	3.59E+00	**3.09E+00**	1.02E+00	**39.99%**	**0.02%**
19	3.37E+00	9.45E-01	3.31E+00	5.08E-01	**1.74%**	71.13%

Estos resultados corroboran los encontrados en la literatura [30], donde se plantea que el modelo local tiende a ser más estable y a reportar mejores resultados al explorar su rendimiento en un conjunto variado de problemas. Es importante aclarar que esto no significa que el modelo local deba ser considerado la opción idónea para todo tipo de problemas; la rápida convergencia del modelo global produce generalmente mejores resultados en problemas unimodales que cualquier topología local debido a que no enfrenta el riesgo de converger prematuramente en óptimos locales. Incluso, bajo ciertas circunstancias (e.g. limitado número de evaluaciones de función), en algunos problemas multimodales muy complejos el modelo global pudiera competir o llegar a obtener mejores soluciones que el modelo local [30]. Esto, y el hecho de que la inicialización uniforme del enjambre en todo el espacio de soluciones puede favorecer el proceso de búsqueda de un algoritmo, podría ser la explicación de por qué, en estos experimentos, la mejor solución encontrada por el modelo global supera a la del modelo local en más de la mitad de las

[2]Prueba estadística utilizada para determinar si la diferencia entre el valor promedio de dos muestras independientes es significativa o no. La hipótesis nula de que las diferencias observadas son casuales se rechaza con un nivel de significación del 5%.

funciones.

Tomando como punto de comparación estos resultados, a continuación se analizarán los efectos que produce la estrategia de reducción de la dimensión a ambas versiones del estándar de PSO.

2.2.5. Análisis de PSO con Reducción de la Dimensión

Los algoritmos descritos al comienzo de este capítulo, insertan la estrategia de la reducción de la dimensión en PSO mediante el uso del parámetro dim_r, que especifica la cantidad de dimensiones que se explorarán en cada período de k iteraciones. Para evaluar los efectos que produce esta estrategia en el comportamiento de PSO, en los experimentos realizados se tuvo en cuenta, además de los parámetros descritos anteriormente, diferentes valores del parámetro $dim_r = 1, 5, 10, 20$ y $k = 50$.

Los algoritmos serán analizados en el mismo orden en que fueron presentados, primero SACR-MPSO, luego SASR-MPSO y por último BE-MPSO. Se usarán como subíndice las letras g y l para referirse a las implementaciones con topología global y local respectivamente.

SACR-MPSO

En las tablas 2.5 y 2.6 se muestra, para cada valor de dim_r el promedio del error absoluto obtenido por SACR-MPSO$_g$ y SACR-MPSO$_l$, respectivamente. Los valores que se resaltan representan mejoras o empates de la variante analizada respecto a la estándar.

Tabla 2.5: Errores absolutos de SACR-MPSO$_g$

fn	$dim_r = 1$	$dim_r = 5$	$dim_r = 10$	$dim_r = 20$	PSO$_g$
1	1.02E+01	9.83E-02	5.37E-01	**0.00E+00**	0.00E+00
2	8.95E+04	2.64E+02	3.01E+03	**9.45E+01**	1.91E+02
3	1.83E+02	**2.69E+01**	5.03E+01	**3.85E+01**	4.12E+01
4	2.38E+02	**3.99E+01**	5.48E+01	5.17E+01	4.84E+01
5	3.00E+01	**9.24E-01**	**5.68E+00**	**7.76E+00**	1.13E+01
15	2.83E+02	1.35E+02	1.01E+02	**6.25E+01**	6.26E+01
16	1.82E+01	1.19E+01	**4.83E+00**	6.35E+00	5.38E+00
17	6.27E+00	2.49E+00	2.40E+00	1.40E+00	1.39E+00
18	2.36E+01	8.90E+00	9.24E+00	5.36E+00	5.15E+00
19	6.79E+00	4.38E+00	3.97E+00	**3.30E+00**	3.37E+00

En las tablas 2.7 y 2.8 se muestran las diferencias porcentuales – calculadas mediante la fórmula $(a - b)/a$, considerando a y b el promedio del error absoluto de PSO estándar y SACR-MPSO, respectivamente – y los

Tabla 2.6: Errores absolutos de SACR-MPSO$_l$

fn	$dim_r = 1$	$dim_r = 5$	$dim_r = 10$	$dim_r = 20$	PSO$_l$
1	1.95E+01	6.00E-04	5.16E-09	**0.00E+00**	0.00E+00
2	3.21E+05	1.00E+01	1.20E-05	**0.00E+00**	0.00E+00
3	2.52E+02	3.50E+01	**2.57E+01**	**3.02E+01**	3.14E+01
4	3.24E+02	4.57E+01	**3.01E+01**	**3.96E+01**	4.10E+01
5	5.03E+01	**0.00E+00**	**0.00E+00**	**0.00E+00**	0.00E+00
15	3.77E+02	1.50E+02	9.65E+01	**5.14E+01**	5.28E+01
16	1.92E+01	1.32E+01	8.28E+00	6.63E+00	6.32E+00
17	8.72E+00	3.16E+00	1.48E+00	7.00E-01	5.74E-01
18	3.20E+01	1.17E+01	5.69E+00	**3.07E+00**	3.09E+00
19	8.92E+00	4.22E+00	3.80E+00	**3.13E+00**	3.31E+00

resultados de los t-tests realizados entre ellos. En las columnas correspondientes a los t-tests, se resaltan los valores que confirman una diferencia significativa entre los dos algoritmos, y a partir de estos, en las columnas correspondientes a las diferencias porcentuales, se resaltan aquellos valores que representan mejoras o empates de SACR-MPSO respecto a la versión estándar de PSO.

Tabla 2.7: SACR-PSO$_g$ vs PSO$_g$

	$dim_r = 1$		$dim_r = 5$		$dim_r = 10$		$dim_r = 20$	
fn	dif-%	t-test	dif-%	t-test	dif-%	t-test	dif-%	t-test
1	$-\infty$	**0.0%**	$-\infty$	26.6%	$-\infty$	**1.4%**	0.0%	97.1%
2	-46725.4%	**0.1%**	-38.2%	60.9%	-1476.7%	**1.3%**	50.6%	43.6%
3	-343.6%	**0.0%**	**34.6%**	**0.0%**	-22.2%	**3.4%**	6.6%	44.2%
4	-392.5%	**0.0%**	**17.4%**	**3.1%**	-13.2%	13.0%	-6.9%	41.1%
5	-164.8%	**0.0%**	**91.9%**	**0.0%**	**49.9%**	**1.9%**	31.6%	15.3%
15	-352.7%	**0.0%**	-115.1%	**0.0%**	-60.7%	**0.0%**	0.1%	98.6%
16	-238.1%	**0.0%**	-121.5%	**0.0%**	10.2%	26.5%	-18.0%	6.4%
17	-352.1%	**0.0%**	-79.4%	**0.0%**	-73.0%	**0.0%**	-0.9%	94.6%
18	-357.9%	**0.0%**	-72.9%	**0.0%**	-79.5%	**0.0%**	-4.1%	74.9%
19	-101.2%	**0.0%**	-29.8%	**0.0%**	-17.8%	**0.0%**	2.1%	67.1%

Haciendo un análisis de estos resultados se puede concluir que:

1. A medida que disminuye el valor de dim_r (mayor reducción de la dimensión) la diferencia entre los resultados de SACR-MPSO y PSO, para ambos modelos de vecindades, se hace mayor.

2. Para $dim_r = 20$ la diferencia entre SACR-MPSO y PSO no es significativa.

3. Las funciones separables (F1-F5) son las más favorecidas con la reduc-

Tabla 2.8: SACR-PSO$_l$ vs PSO$_l$

	$dim_r = 1$		$dim_r = 5$		$dim_r = 10$		$dim_r = 20$	
fn	dif-%	t-test	dif-%	t-test	dif-%	t-test	dif-%	t-test
1	$-\infty$	**0.0%**	$-\infty$	**0.1%**	$-\infty$	**2.1%**	0.0%	44.9%
2	$-\infty$	**2.6%**	$-\infty$	**3.5%**	$-\infty$	**0.0%**	0.0%	7.0%
3	-701.8%	**0.0%**	-11.5%	6.0%	**18.3%**	**0.0%**	3.9%	48.8%
4	-689.7%	**0.0%**	-11.6%	**3.7%**	**26.7%**	**0.0%**	3.3%	48.7%
5	$-\infty$	**0.0%**	0.0%	50.1%	0.0%	48.7%	0.0%	53.0%
15	-613.1%	**0.0%**	-183.1%	**0.0%**	-82.6%	**0.0%**	2.8%	50.4%
16	-203.9%	**0.0%**	-108.6%	**0.0%**	-31.0%	**0.0%**	-4.8%	26.8%
17	-1419.6%	**0.0%**	-450.0%	**0.0%**	-158.6%	**0.0%**	-22.0%	5.4%
18	-934.7%	**0.0%**	-277.2%	**0.0%**	-84.2%	**0.0%**	0.5%	92.0%
19	-169.2%	**0.0%**	-27.5%	**0.0%**	-14.6%	**0.0%**	5.6%	9.7%

ción de la dimensión.

4. En general, la reducción de la dimensión, con mecanismo de selección aleatoria con reemplazo, no produce grandes beneficios al proceso de búsqueda de PSO.

La primera conclusión es de esperar si se considera que $dim_r = d$ significa que en el período actual solo se están explorando (modificando) d dimensiones. Al aumentar la reducción de la dimensión, la cantidad de dimensiones que son exploradas es menor lo cual trae como consecuencia que se produzca un desbalance, en desfavor de la exploración, en el proceso de búsqueda de PSO. A partir de esta relación puede entonces explicarse la segunda conclusión. Dado que $D = 20$, entonces $dim_r = 20$ significa que no hubo reducción de la dimensión y por tanto el comportamiento de PSO y SACR-MPSO es el mismo, luego sus resultados deberían ser similares.

Las funciones separables tienen como característica que la búsqueda del óptimo global puede realizarse efectuando la búsqueda del valor óptimo para cada una de las dimensiones. Precisamente, la inclusión en estos experimentos del conjunto de funciones separables F1-F5 tuvo como objetivo evaluar la capacidad de la reducción de la dimensión como estrategia para explotar la separabilidad de una función. De acuerdo a los resultados alcanzados, puede observarse que los pocos casos en los que SACR-MPSO supera a PSO, ya sea la variante global o local, se corresponden efectivamente con las funciones separables.

Con respecto a la última conclusión, se puede observar que, en general, para ambos modelos de vecindades, el rendimiento de SACR-MPSO es inferior comparado con el del estándar de PSO, lo cual se hace más notable a medida que disminuye el valor de dim_r. Parece ser que, además del número

de dimensiones a considerar en cada período de exploración, un factor con gran influencia en el buen desempeño del algoritmo es el mecanismo utilizado para seleccionar el conjunto de dimensiones DIM_r.

Para el mecanismo de selección aleatoria y con reemplazo utilizado en esta variante, la probabilidad de que una dimensión cualquiera d sea elegida para su exploración en uno de los períodos es igual a $P_s = \frac{dim_r}{D}$ y de que no sea elegida $1 - P_s$. De modo que, la probabilidad de que una dimensión d no sea seleccionada en ninguna de las selecciones de DIM_r es igual a:

$$P_{ns} = \left(1 - \frac{dim_r}{D}\right)^{\frac{n}{k}} \tag{2.2}$$

Donde $\frac{n}{k}$ representa la cantidad de períodos de exploración y por tanto el número de veces que se reelige el conjunto de dimensiones DIM_r.

De esta ecuación se puede inferir que a medida que aumenta el valor de dim_r menor será la probabilidad de que una dimensión no sea seleccionada nunca; y mientras mayor sea la cantidad de dimensiones exploradas en todo el proceso mayor será la calidad de las soluciones encontradas. También se cumple que, aumentando el número de iteraciones n disminuye P_{ns} y aumentando la cantidad de iteraciones por período k aumenta P_{ns}. En este sentido, es importante una buena selección tanto de n como de k. Es necesario un k no tan pequeño de modo que se pueda asegurar una mayor exploración de las dimensiones seleccionadas y no tan grande respecto al valor de n de modo que mayor cantidad de dimensiones sean seleccionadas durante todo el proceso. De no existir un límite de recursos y tiempo, la configuración ideal para esta variante sería tomar n y k bien grandes, n mucho más grande que k para asegurar una adecuada exploración de todas las dimensiones.

SASR-MPSO

Como se describe en la sección 2.1.2 el mecanismo de selección en esta variante es aleatorio, sin embargo, no se realiza sobre todo el conjunto D como en la variante anterior, sino sobre el conjunto de dimensiones que no han sido exploradas todavía. Este conjunto R va disminuyendo su cardinalidad a medida que las dimensiones van siendo seleccionadas y cuando llega a 0 (no queda ninguna por seleccionar) se reinicia con todas las dimensiones de D. De modo que, si $\frac{n}{k} \geq \frac{D}{dim_r}$ se asegura que todas las dimensiones serán exploradas y si $\frac{n}{k} = p * \frac{D}{dim_r}$, entonces, todas lo serán al menos p veces. Bajo este mecanismo de selección se espera que esta variante sea más exploratoria y por tanto que produzca mejores resultados que la anterior.

Puesto que para $dim_r = 20$ el algoritmo mantiene un comportamiento similar al estándar de PSO, se decidió evaluar solo los resultados de esta variante para $dim_r = 1, 5, 10$. En las tablas 2.9, 2.10, 2.11 y 2.12 pueden encontrarse los resultados para SASR-MPSO$_g$ y SASR-MPSO$_l$. En ellas se resaltan los errores absolutos iguales o inferiores a los de la versión estándar y las diferencias porcentuales, pero estas solo cuando las diferencias son significativas.

Tabla 2.9: Errores absolutos de SASR-MPSO$_g$

fn	$dim_r = 1$	$dim_r = 5$	$dim_r = 10$	PSO$_g$
1	2.05E-01	2.30E-02	2.30E-02	0.00E+00
2	9.23E+02	**9.23E+01**	7.84E+02	1.91E+02
3	7.74E+01	**2.12E+01**	**3.11E+01**	4.12E+01
4	9.74E+01	**2.75E+01**	**4.21E+01**	4.84E+01
5	**4.11E-01**	**0.00E+00**	**2.43E+00**	1.13E+01
15	2.23E+02	9.15E+01	9.32E+01	6.26E+01
16	1.78E+01	9.52E+00	5.79E+00	5.38E+00
17	4.91E+00	2.03E+00	2.31E+00	1.39E+00
18	1.81E+01	7.40E+00	6.83E+00	5.15E+00
19	5.68E+00	4.17E+00	4.02E+00	3.37E+00

Tabla 2.10: Errores absolutos de SASR-MPSO$_l$

fn	$dim_r = 1$	$dim_r = 5$	$dim_r = 10$	PSO$_l$
1	5.54E-01	3.02E-07	**0.00E+00**	0.00E+00
2	1.36E+03	7.84E-04	**0.00E+00**	0.00E+00
3	1.16E+02	**3.05E+01**	**2.30E+01**	3.14E+01
4	1.64E+02	**3.57E+01**	**3.41E+01**	4.10E+01
5	1.56E-01	**0.00E+00**	**0.00E+00**	0.00E+00
15	2.89E+02	1.32E+02	8.15E+01	5.28E+01
16	1.83E+01	1.17E+01	7.56E+00	6.32E+00
17	7.24E+00	2.46E+00	1.05E+00	5.74E-01
18	2.60E+01	9.19E+00	4.59E+00	3.09E+00
19	7.31E+00	4.22E+00	3.79E+00	3.31E+00

Como se puede observar, para ambos modelos la cantidad de mejoras respecto al promedio del error absoluto duplicó las obtenidas por la variante con reemplazo. Según los tests estadísticos de las tablas 2.11 y 2.12, no todas estas mejoras son significativas, pero en su gran mayoría superan las obtenidas por la variante anterior. Considerando estos resultados, parece ser que el mecanismo utilizado para seleccionar las dimensiones a explorar en cada período sí juega un papel importante. El mecanismo de selección aleatoria sin reemplazo no es que sea el más eficiente de todos pero al menos

asegura que ninguna dimensión quede sin explorar, lo cual no se garantiza por la variante aleatoria con reemplazo. En este sentido, sería interesante experimentar con otros mecanismos que aseguren la exploración de todas las dimensiones y que de alguna forma, en caso de que existan dimensiones correlacionadas, se percate de ello para explorarlas en un mismo período, en lugar de hacer la selección de manera aleatoria.

Tabla 2.11: SASR-MPSO$_g$ vs PSO$_g$

	$dim_r = 1$		$dim_r = 5$		$dim_r = 10$	
fn	dif-%	t-test	dif-%	t-test	dif-%	t-test
1	$-\infty$	**0.3%**	$-\infty$	32.2%	$-\infty$	32.2%
2	-382.9%	**0.9%**	51.7%	35.2%	-310.3%	5.1%
3	-88.0%	**0.0%**	**48.6%**	**0.0%**	**24.4%**	**0.5%**
4	-101.4%	**0.0%**	**43.1%**	**0.0%**	13.0%	14.2%
5	**96.4%**	**0.0%**	**100.0%**	**0.0%**	**78.6%**	**0.0%**
15	-255.9%	**0.0%**	-46.2%	**0.0%**	-48.8%	**0.0%**
16	-231.4%	**0.0%**	-77.1%	**0.0%**	-7.7%	38.9%
17	-253.9%	**0.0%**	-46.2%	**0.2%**	-66.3%	**0.0%**
18	-251.2%	**0.0%**	-43.8%	**0.0%**	-32.7%	**4.4%**
19	-68.5%	**0.0%**	-23.7%	**0.0%**	-19.2%	**0.0%**

Tabla 2.12: SASR-MPSO$_l$ vs PSO$_l$

	$dim_r = 1$		$dim_r = 5$		$dim_r = 10$	
fn	dif-%	t-test	dif-%	t-test	dif-%	t-test
1	$-\infty$	**0.0%**	$-\infty$	**0.0%**	0.0%	56.4%
2	$-\infty$	**0.0%**	$-\infty$	**0.0%**	0.0%	81.2%
3	-269.8%	**0.0%**	2.9%	57.6%	**26.6%**	**0.0%**
4	-299.7%	**0.0%**	**12.9%**	**0.9%**	**16.7%**	**0.1%**
5	$-\infty$	**1.5%**	0.0%	50.0%	0.0%	50.0%
15	-447.7%	**0.0%**	-150.7%	**0.0%**	-54.2%	**0.0%**
16	-190.0%	**0.0%**	-85.8%	**0.0%**	-19.6%	**0.1%**
17	-1161.9%	**0.0%**	-328.2%	**0.0%**	-83.5%	**0.0%**
18	-742.9%	**0.0%**	-197.4%	**0.0%**	-48.7%	**0.0%**
19	-120.6%	**0.0%**	-27.3%	**0.0%**	-14.2%	**0.0%**

Otra observación que llama la atención al examinar los resultados es que ninguna de las mejoras pertenecen al conjunto de funciones multimodales y, además, los resultados obtenidos para estas funciones son significativamente peores que los obtenidos por la versión estándar. Al parecer, la estrategia de reducir la dimensión, incluso empleando un mecanismo de selección que asegura la exploración de todas las dimensiones, no es favorable para estas funciones en la misma forma en que lo es para las funciones separables.

Los experimentos realizados con Locust Swarms en [44] muestran que

la reducción de la dimensión (e.g $dim_r = 5$) para este mismo conjunto de funciones puede mejorar significativamente los resultados obtenidos por el estándar de PSO. Aparentemente estos resultados son contradictorios. Sin embargo, en Locust Swarms la reducción de la dimensión solo se utiliza en la fase de creación de los exploradores, y el proceso de búsqueda es llevado a cabo por un PSO con características muy similares al del estándar. En las variantes implementadas de MPSO, por su parte, la reducción de la dimensión forma parte del proceso de búsqueda del algoritmo haciéndolo menos exploratorio que el de Locust Swarms y aún más que el de PSO. En este sentido, es de esperar que sus resultados no sean muy buenos en funciones que requieren de mecanismos más eficientes de exploración.

BE-MPSO

A diferencia de las variantes analizadas anteriormente, BE-MPSO no posee un mecanismo de selección, pues todas las combinaciones de dim_r dimensiones en D son exploradas en cada iteración. Debido al alto costo computacional que implica hacer una búsqueda exhaustiva, en cada iteración, por todas las posibles combinaciones de dim_r dimensiones en D, los experimentos con BE-MPSO solo se realizaron para $dim_r = 1$. Nótese que el costo de esta variante, medido en términos de la cantidad de posiciones de evaluación que se generan y evalúan en todo el proceso, es equivalente a:

$$FE = n * S * \binom{D}{dim_r}$$

Donde D es la dimensión del espacio, dim_r la cantidad de dimensiones a explorar en cada período, S la cantidad de partículas que componen el enjambre y n la cantidad de iteraciones que el algoritmo realiza. Esto implica que con $D = 20$, $S = 50$ y $n = 2000$ (número máximo de iteraciones que realizan los algoritmos anteriores), la cantidad de evaluaciones de función que realizaría BE-MPSO sería igual a $PE = 2000000$, 20 veces superior a la cantidad que se realizaron en los experimentos anteriores.

Este es un ejemplo claro donde se evidencia el por qué se ha tomado como estándar comparar el rendimiento entre dos o más algoritmos respecto al número de evaluaciones de función y no respecto al número de iteraciones. La razón fundamental es que en una iteración pueden realizarse tantas evaluaciones de función (solución) como se quiera y esto influye directamente en el rendimiento del algoritmo, ya que mientras más grande sea el número de soluciones que se evalúen (exploren) mayor será la probabilidad de encontrar la solución óptima o una cercana a ella. En este sentido, si se compararan

los algoritmos PSO estándar y BE-MPSO fijando el número de iteraciones, BE-MPSO tendría más posibilidades de encontrar una mejor solución, pues en cada iteración explora por cada partícula 19 soluciones más que la versión estándar.

Con el objetivo de realizar una comparación justa entre estos dos algoritmos, se fijó el número de evaluaciones de función en $FE = 2000000$, lo cual implica que la versión estándar realizó 20 veces más iteraciones ($n = 40000$) que BE-MPSO. Los resultados obtenidos en estos experimentos se muestran en las tablas 2.13 y 2.14.

Los resultados de la implementación global de BE-MPSO para las funciones separables son, en su totalidad, significativamente mejores que los obtenidos por PSO estándar. Esto confirma una vez más que la reducción de la dimensión puede ser favorable para las funciones separables. Para las no separables las diferencias son significativas y negativas lo cual significa que no se obtuvo ninguna mejora con respecto al estándar.

Tabla 2.13: Resultados de BE-MPSO$_g$ para $dim_r = 1$

fn	Error abs.	Error rel.	dif-%	t-test	PSO$_g$
1	**0.00E+00**	0.0%	0.0%	**2.7%**	0.00E+00
2	**1.23E-06**	0.0%	100.0%	**4.4%**	3.74E+02
3	**3.27E-05**	0.0%	100.0%	**0.0%**	4.04E+01
4	**4.08E-04**	0.0%	100.0%	**0.0%**	5.31E+01
5	**0.00E+00**	0.0%	100.0%	**0.0%**	1.13E+01
15	1.32E+02	13.2%	-100.6%	**0.0%**	6.58E+01
16	6.18E+00	8.7%	-16.3%	**1.7%**	5.32E+00
17	4.26E+00	25.2%	-199.0%	**0.0%**	1.43E+00
18	1.48E+01	87.5%	-172.2%	**0.0%**	5.45E+00
19	3.31E+00	3.2%	-140.7%	**0.0%**	1.37E+00

Tabla 2.14: Resultados de BE-MPSO$_l$ para $dim_r = 1$

fn	Error abs.	Error rel.	dif-%	t-test	PSO$_l$
1	2.77E-09	0.0%	0.0%	**0.0%**	0.00E+00
2	7.39E-06	0.0%	0.0%	**0.0%**	0.00E+00
3	**5.76E-04**	0.0%	100.0%	**0.0%**	2.42E+01
4	**3.50E-03**	0.0%	100.0%	**0.0%**	3.29E+01
5	0.00E+00	0.0%	0.0%	65.8%	0.00E+00
15	1.82E+02	18.2%	-350.4%	**0.0%**	4.05E+01
16	8.29E+00	11.6%	-58.7%	**0.0%**	5.22E+00
17	5.95E+00	35.2%	-1475.0%	**0.0%**	3.78E-01
18	2.08E+01	122.7%	-902.0%	**0.0%**	2.07E+00
19	4.17E+00	4.1%	-113.1%	**0.0%**	1.96E+00

En el caso de la variante local de BE-MPSO, se puede observar que el aumento del número de evaluaciones de función favorece al algoritmo estándar lo cual confirma el hecho de que su velocidad de convergencia es lenta y que sus resultados pueden mejorar considerablemente si se le da la posibilidad de evaluar más soluciones. No obstante, se señala que para dos de las funciones separables, la variante con reducción de la dimensión proporcionó mejores resultados, en una función se alcanzó el óptimo y en las otras dos el error estuvo muy cercano a cero.

Conclusiones

En esta sección se analizó la estrategia de Reducción de la Dimensión y el efecto que produce en la versión estándar de PSO. Para ello se evaluaron tres variantes del algoritmo MPSO que implementan diferentes mecanismos para reducir el número de dimensiones consideradas durante el proceso de exploración en PSO. De acuerdo a los resultados obtenidos por estos algoritmos se puede concluir que: (1) la reducción de la dimensión mejora el rendimiento de PSO estándar en funciones separables y (2) un factor que influye decisivamente en el rendimiento del algoritmo es el mecanismo que se utilice para determinar qué dimensiones explorar en cada momento.

A partir de los experimentos realizados con las dos primeras variantes de MPSO se pudo comprobar que un mecanismo que garantiza la exploración de todas las dimensiones produce mejores resultados que uno que no asegura tal condición. Con la variante BE-MPSO se mostró, además, que la selección adecuada de las dimensiones a explorar en cada momento influye también en los resultados obtenidos.

La implementación global de BE-MPSO superó los resultados de la estándar para todas las funciones separables (con un error relativo del 0,0 %), pero a un costo computacional muy grande. La duración de todo el experimento con esta variante (50 ejecuciones por 10 funciones) demoró, en una Pentium (R) Dual-Core CPU T4500 a 2.30GHz y 3GB de RAM, aproximadamente 8 horas. Considerando que la cantidad de evaluaciones de función que realiza esta variante por iteración dependen de la cantidad de combinaciones de dim_r dimensiones en D, los experimentos para otros valores de $dim_r > 1$ resultaron impracticables. Obsérvese que solo incrementando dim_r en 1 el valor de FE aumenta a 19000000, casi 10 veces la cantidad utilizada para $dim_r = 1$. En este sentido, se podría decir que las aplicaciones prácticas de esta variante son reducidas. No obstante a estas limitantes, los resultados obtenidos pueden servir de motivación para explorar otros mecanismos de selección no aleatorios ni exhaustivos, que permitan obtener resultados

similares realizando menor cantidad de evaluaciones de función.

Capítulo 3

Optimización de Enjambre de Partículas con Retorno y Actualización Retardada

En este capítulo se introducen dos estrategias: *Retorno* y *Actualización Retardada* para incrementar, respectivamente, la capacidad de explotar y explorar el espacio de búsqueda en el algoritmo PSO. Las dos variantes de PSO con Retorno que se proponen están inspiradas en el proceso de intensificación que realizan algunas metaheurísticas como Búsqueda Tabú. La Actualización Retardada se aplica como una estrategia de diversificación cuya efectividad se debe a la prolongación de la fase de exploración en el algoritmo.

3.1. Estrategia de Retorno

Como se ha reportado en la literatura [37, 30], el modelo global (topología de estrella) tiene la ventaja de converger rápidamente mientras que al modelo local (topología de anillo con $n = 2$) lo favorece un proceso de exploración más extenso. La rápida convergencia hace al modelo global más sensible a una convergencia prematura en un óptimo local, mientras que la posibilidad de explorar en paralelo diferentes áreas del espacio de búsqueda y la comunicación parcial entre las partículas hacen que el modelo local sea menos vulnerable a un estancamiento en la región de un óptimo local pero al costo de una convergencia más lenta. En este sentido, se puede decir que el modelo global se caracteriza por ser más de explotación y el modelo local de más

exploración.

Dado que la principal diferencia entre estos dos modelos es la red de comunciación social que las partículas usan para intercambiar experiencias, un método simple para mejorar el balance entre exploración y explotación en PSO podría ser usar una topología de vecindades intermedia entre ellos que combine sus ventajas. Un procedimiento lógico sería experimentar variando el número de vecinos de cada partícula hasta encontrar el valor óptimo. El problema con este enfoque es que, a medida que la red de comunicación se va acercando al modelo global, la convergencia se hace más rápida pero al costo de incrementar la vulnerabilidad a estancarse en óptimos locales. Por el contrario, si la solución se acerca al modelo local, su capacidad de exploración se hace mayor pero al costo de una convergencia más lenta. En este sentido, buscar una topología de vecindades óptima implica resolver un problema de optimización multi-objetivo con objetivos opuestos y para este tipo de problemas no existe una única solución óptima sino un conjunto de soluciones Pareto Optimales, de las cuales ninguna solución puede decirse que es la mejor de todas porque son soluciones *no dominadas*[1] [50].

Otros enfoques podrían ser comenzar con un proceso muy explorativo e ir disminuyendo poco a poco la exploración hasta concentrar la búsqueda en una región del espacio o combinar varias fases de este proceso. Estas ideas han sido implementadas con éxito en otras técnicas de búsqueda como en Recocido Simulado [7, 8] a través de su mecanismo de ajuste de la temperatura y el proceso de enfriamiento y calentamiento (cooling and reheating) [51]). El problema con estos enfoques es la dificultad que implica la selección de un mecanismo eficiente para pasar de exploración a explotación y determinar el mejor momento para comenzar una nueva fase de exploración – explotación.

Para eludir estos problemas muchos algoritmos prefieren otra forma de balancear explotación y exploración: el uso de un mecanismo de reinicio o de múltiples procesos de optimización local similar al que realiza Algoritmos Meméticos [46]. La idea consiste en realizar una búsqueda local hasta satisfacer ciertas condiciones (e.g. estancamiento en un óptimo local, disminución de la diversidad, etc.) y luego continuar la búsqueda reiniciando el algoritmo en regiones no exploradas. Los mecanismos de reinicio simples tienen como inconveniente que no ofrecen garantía de recomenzar la búsqueda en una región del espacio sin explorar.

[1] Dado un problema de optimización de M funciones objetivos, una solución x_1 se dice que *domina* a otra solución x_2 si se cumplen simultáneamente: (i) la solución x_1 no es peor que x_2 en todas las funciones objetivos y (ii) la solución x_1 es estrictamente mejor que x_2 en al menos una función objetivo. Si alguna de las condiciones anteriores no se cumple, se dice entonces que la solución x_1 *no domina* a x_2.

A diferencia de un reinicio "ciego" como el anterior, la *estrategia de retorno* que se propone concentra todo el esfuerzo del proceso de búsqueda en las regiones más prometedoras del espacio. Con cierta semejanza a la fase de intensificación que se realiza en Búsqueda Tabú, la estrategia de retorno provoca que el enjambre de partículas regrese a las regiones más promisorias, descubiertas en previas iteraciones, para buscar en ellas exhaustivamente. El procedimiento se inserta en un PSO estándar añadiendo una fase de intensificación de la búsqueda cada k iteraciones, esto es, las partículas retornan a las mejores posiciones (*pbest*) encontradas individualmente durante las últimas k iteraciones (que podría ser la posición inicial de no encontrarse una mejor posición durante estas iteraciones). Dos variantes surgen de este procedimiento: *PSO con Retorno* (R-PSO) y *PSO con Retorno y Exploración* (RE-PSO).

3.1.1. PSO con Retorno (R-PSO)

En la variante de PSO con Retorno el regreso de las partículas hacia las mejores posiciones encontradas individualmente en previas iteraciones es independiente de si en las últimas k iteraciones la partícula no ha encontrado una mejor solución.

La implementación de esta fase de intensificación se apoya en la capacidad que tienen las partículas para recordar sus éxitos más recientes mediante la variable *pbest*. El proceso de actualización de las velocidades y las posiciones de las partículas (Algoritmo 7) no sufre transformaciones respecto al PSO estándar, la diferencia radica en que al final de las k iteraciones las partículas retornan a su *pbest* (pasos 9-13).

Algoritmo 7 PSO con Retorno (R-PSO)

1. Inicializar constantes y variables (D, S, k, T, x_i^0, v_i^0, ...)
2. **for** $t = 1$ to T **do**
3. **for** $i = 1$ to S **do**
4. Actualizar v_i^t usando ecuación (1.12)
5. Actualizar x_i^t usando ecuación (1.6)
6. Actualizar $pbest_i^t$ usando ecuación (1.7)
7. **end for**
8. Actualizar $gbest_i^t$ usando ecuación (1.8)
9. **if** $t \equiv 0$ (mód k) **then**
10. **for** $i = 1$ to S **do**
11. $x_i^t \leftarrow pbest_i^t$
12. **end for**
13. **end if**
14. **end for**

Una consecuencia significativa del retorno sin restricciones es que puede limitar la exploración del espacio de búsqueda. Si una partícula ha quedado atrapada en la región de un óptimo local será más difícil que logre escapar si cada cierta cantidad de iteraciones se le obliga a retornar a su mejor posición histórica. Este comportamiento se hará más probable en el modelo global de PSO, donde el movimiento de todo el enjambre es guiado por una sola partícula líder. En un escenario como este, las demás partículas del enjambre se verán atraídas rápidamente por la posición *gbest* siendo inminente la convergencia del enjambre en la región. El modelo local de PSO, aunque no está ajeno a este comportamiento, será menos vulnerable a sus efectos al verse favorecido de una mayor exploración del espacio de búsqueda por tener más de una partícula líder guiando simultáneamente el movimiento de las demás.

Otra posible consecuencia de esta estrategia es el refinamiento de las mejores posiciones encontradas por el enjambre. Para una partícula, una vez atrapada en la región de un óptimo local, la estrategia de retorno puede producir un efecto similar al de una estrategia de búsqueda local. Tras reiniciar el proceso de búsqueda en su mejor posición histórica, la partícula realiza, durante k iteraciones, una búsqueda exhaustiva en la región con posibilidades de encontrar una mejor posición si aún no ha convergido al óptimo local. Este comportamiento produciría entonces más beneficios después de explorar eficientemente el espacio de búsqueda e identificar las regiones más prometedoras.

3.1.2. PSO con Retorno y Exploración (RE-PSO)

En la variante de PSO con Retorno y Exploración las partículas realizan el retorno solo si en las últimas k iteraciones la mejor posición encontrada individualmente ha cambiado, es decir, si se ha encontrado una mejor posición. Si esto no se cumpliera las partículas son libres de continuar su camino.

Para la implementación de la fase de intensificación de esta variante no es suficiente con que las partículas recuerden su mejor posición histórica (*pbest*) sino que requieren de otra memoria explícita de almacenamiento – *pkbest* – que les permita recordar los éxitos logrados hasta la última fase de exploración. En el algoritmo 8 (pasos 14 y 15) se muestra que el retorno está condicionado a la evaluación de las posiciones *pbest* y *pkbest*, llevándose a cabo si *pbest* resulta ser mejor que *pkbest*.

A diferencia de la fase de intensificación del R-PSO que puede ocurrir múltiples veces la del RE-PSO solo se realiza una vez sobre una misma

Algoritmo 8 PSO con Retorno y Exploración (RE-PSO)

1. Inicializar constantes y variables (D, S, k, T, x_i^0, v_i^0, ...)
2. **for** $i = 1$ to S **do**
3. $pkbest_i \leftarrow pbest_i^0$
4. **end for**
5. **for** $t = 1$ to T **do**
6. **for** $i = 1$ to S **do**
7. Actualizar v_i^t usando ecuación (1.12)
8. Actualizar x_i^t usando ecuación (1.6)
9. Actualizar $pbest_i^t$ usando ecuación (1.7)
10. **end for**
11. Actualizar $gbest_i^t$ usando ecuación (1.8)
12. **if** $t \equiv 0$ (mód k) **then**
13. **for** $i = 1$ to S **do**
14. **if** $f(pbest_i^t) < f(pkbest_i)$ **then**
15. $x_i^t \leftarrow pbest_i^t$
16. **end if**
17. $pkbest_i \leftarrow pbest_i^t$
18. **end for**
19. **end if**
20. **end for**

posición. Si las partículas no mejoran su mejor posición individual en las últimas k iteraciones, ante la posibilidad de quedar atrapadas en la región de un óptimo local, se favorece la exploración de nuevas áreas del espacio de búsqueda evitando el retorno. Se espera por tanto que el algoritmo RE-PSO tenga un comportamiento más exploratorio que R-PSO.

3.2. Estrategia de Actualización Retardada (AR-PSO)

Otra forma de alterar el balance entre la exploración y la explotación de PSO es con la *actualización retardada*, cada k iteraciones, de las mejores posiciones encontradas individualmente (*pbest*) y las encontradas por las partículas vecinas (*gbest*). Normalmente, la actualización de estas posiciones se realiza *on-line*, es decir, después de que las partículas han realizado un nuevo movimiento, de manera que la experiencia de cada una podrá beneficiar a las demás de inmediato. Consecuencia del retraso que se produce en la actualización de *pbest* y *gbest* es el incremento en la exploración de las regiones alrededor de estas posiciones.

La implementación de este procedimiento puede realizarse muy fácilmente (Algoritmo 9) y solo requiere una variable extra por cada partícula

para almacenar los éxitos logrados individualmente durante las k iteraciones en transcurso. En esencia, cada partícula tendrá dos memorias locales: *pkbest* y *pbest*. La posición *pkbest* se actualiza en cada iteración utilizando la ecuación (3.1) y su función es almacenar la mejor posición encontrada durante las k iteraciones del período actual. La posición *pbest* mantiene la función de guiar el movimiento de las partículas en cada iteración, pero se actualiza al final de cada período en caso de que la posición *pkbest* evalúe la función objetivo mejor que ella, es decir, si se ha encontrado una mejor posición en el período que acaba de completarse.

$$pkbest_i^{t+1} = \begin{cases} pkbest_i^t & \text{si} \quad f(x_i^{t+1}) \geq f(pkbest_i^t) \\ x_i^{t+1} & \text{si} \quad f(x_i^{t+1}) < f(pkbest_i^t) \end{cases} \tag{3.1}$$

Algoritmo 9 PSO con Actualización Retardada (AR-PSO)

1. Inicializar constantes y variables (D, S, k, T, x_i^0, v_i^0, ...)
2. **for** $i = 1$ to S **do**
3. $\quad pkbest_i^0 \leftarrow pbest_i$
4. **end for**
5. **for** $t = 1$ to T **do**
6. $\quad$ **for** $i = 1$ to S **do**
7. $\quad\quad$ Actualizar v_i^t usando ecuación (3.2)
8. $\quad\quad$ Actualizar x_i^t usando ecuación (1.6)
9. $\quad\quad$ Actualizar $pkbest_i^t$ usando ecuación (3.1)
10. $\quad$ **end for**
11. $\quad$ **if** $t \equiv 0$ (mód k) **then**
12. $\quad\quad$ **for** $i = 1$ to S **do**
13. $\quad\quad\quad pbest_i \leftarrow pkbest_i^t$
14. $\quad\quad$ **end for**
15. $\quad\quad$ **for** $i = 1$ to S **do**
16. $\quad\quad\quad$ Actualizar $gbest_i$ usando ecuación (1.8)
17. $\quad\quad$ **end for**
18. $\quad$ **end if**
19. **end for**

Nótese que en la ecuación (3.2), que se emplea para actualizar las velocidades, la letra t utilizada como superíndice sobre las posiciones *pbest* y *gbest*, para indicar su valor en cada iteración, se ha eliminado de la ecuación, puesto que estas no se actualizan en cada iteración del algoritmo.

$$v_{i,j}^{t+1} = \chi\left(v_{i,j}^t + c_1 r_{1_{i,j}}^t [pbest_{i,j} - x_{i,j}^t] + c_2 r_{2_{i,j}}^t [gbest_{i,j} - x_{i,j}^t]\right) \tag{3.2}$$

3.3. Estrategias de Retorno y Actualización Retardada combinadas

La estrategia de actualización retardada puede combinarse fácilmente con las dos formas derivadas de la estrategia de retorno. Ambos mecanismos se llevan a cabo cada k iteraciones, de modo que, pueden realizarse simultáneamente. La implementación de estos procedimientos puede hacerse tomando como base las instrucciones de R-PSO (Algoritmo 7) y RE-PSO (Algoritmo 8), añadiéndole a cada uno la actualización de *gbest* luego de actualizado *pbest* y efectuado el retorno.

Como ejemplo se muestra en el algoritmo 10 la combinación de PSO con Retorno con Actualización Retardada.

Algoritmo 10 PSO con Retorno y Actualización Retardada (RAR-PSO)

1. Inicializar constantes y variables (D, S, k, T, x_i^0, v_i^0, ...)
2. **for** $i = 1$ to S **do**
3. $pkbest_i^0 \leftarrow pbest_i$
4. **end for**
5. **for** $t = 1$ to T **do**
6. **for** $i = 1$ to S **do**
7. Actualizar v_i^t usando ecuación (1.12)
8. Actualizar x_i^t usando ecuación (1.6)
9. Actualizar $pkbest_i^t$ usando ecuación (3.1)
10. **end for**
11. **if** $t \equiv 0$ (mód k) **then**
12. **for** $i = 1$ to S **do**
13. $pbest_i \leftarrow pkbest_i^t$
14. $x_i^t \leftarrow pbest_i$
15. **end for**
16. **for** $i = 1$ to S **do**
17. Actualizar $gbest_i$ usando ecuación (1.8)
18. **end for**
19. **end if**
20. **end for**

Siendo el retorno una estrategia de explotación y la actualización retardada una que favorece la exploración entonces combinándolas se podría obtener una técnica que favorezca tanto la explotación como la exploración del espacio de soluciones.

3.4. Resultados Experimentales

En esta sección se describen los experimentos realizados para analizar los efectos que producen las estrategias de Retorno y Actualización Retardada en PSO, tomando como caso de estudio los algoritmos descritos en este capítulo y la versión estándar de PSO presentada en la sección 1.4 del capítulo 1. De manera similar a los experimentos realizados con la estrategia de Reducción de la Dimensión, se consideró la implementación de dos versiones por cada algoritmo, una con topología de estrella (modelo global) y la otra con topología de anillo (modelo local con $n = 2$) para un total de 10 algoritmos. El análisis y comparación de cada una de estas implementaciones se realiza respecto a los resultados obtenidos por la versión global y local de PSO estándar presentados en el capítulo anterior (sección 2.2.4).

3.4.1. Entorno de desarrollo

Como entorno de desarrollo se utilizó Matlab por las mismas razones expuestas en el capítulo anterior (sección 2.2.1).

3.4.2. Funciones de prueba

Los experimentos se realizaron sobre una parte del conjunto de funciones de prueba BBOB [48], en particular, sobre el conjunto de funciones multimodales F15-F19, debido a que sus espacios de búsqueda requieren tanto de técnicas de exploración (multimodalidad) como de explotación (estructura global). No se utiliza el conjunto de funciones separables F1-F5 debido a que las estrategias que se analizan no están dirigidas a explotar la separabilidad de una función.

3.4.3. Diseño de los experimentos

Se tomó como base el diseño de experimentos descrito en la sección 2.2.3, al que se añadió el parámetro k debido a que los algoritmos analizados lo utilizan para especificar cada cuántas iteraciones se hace efectivo el retorno o se actualizan las posiciones *pbest* y *gbest*. Como era de interés evaluar el comportamiento de cada estrategia ante la variación de este parámetro los experimentos se realizaron para $k = 1, 10, 50, 100, 1000$.

3.4.4. Análisis de PSO con Retorno

En las tablas 3.1 y 3.2 se muestra el promedio del error absoluto para cada uno de los valores de k, correspondientes a los resultados obtenidos por las versiones global y local de los algoritmos R-PSO y RE-PSO. En cada caso, se resaltan los resultados que superan a los obtenidos por las variantes del algoritmo estándar, cuyos resultados se muestran en detalle en la sección 2.2.4. En las tablas 3.3 y 3.4 puede observarse cuáles de estas mejoras son significativas.

Después de analizar los resultados resumidos en estas tablas se puede concluir que:

1. Los resultados obtenidos por R-PSO para $k = 1$ son significativamente inferiores a los obtenidos por el estándar, contrario a RE-PSO para $k = 1$ cuyos resultados no difieren esencialmente de los del estándar.

2. Los efectos negativos del retorno, con o sin exploración, son menores en el modelo local que en el global.

3. En general, la estrategia de retorno no aporta mejoras significativas para ninguno de los dos modelos de vecindades.

Tabla 3.1: Errores absolutos de R-PSO$_g$ y RE-PSO$_g$

Algoritmo	fn	$k = 1$	$k = 10$	$k = 50$	$k = 100$	$k = 1000$
R-PSO$_g$	15	2.77E+02	6.98E+01	6.27E+01	**6.11E+01**	**6.02E+01**
	16	1.76E+01	6.30E+00	6.02E+00	5.96E+00	6.07E+00
	17	6.71E+00	1.73E+00	**1.25E+00**	1.40E+00	1.41E+00
	18	2.03E+01	5.94E+00	5.85E+00	5.62E+00	**4.13E+00**
	19	1.01E+01	**3.13E+00**	3.47E+00	**3.18E+00**	3.41E+00
RE-PSO$_g$	15	6.38E+01	6.73E+01	6.42E+01	6.72E+01	**6.26E+01**
	16	**5.33E+00**	5.59E+00	5.72E+00	5.92E+00	6.25E+00
	17	1.70E+00	1.67E+00	1.62E+00	1.85E+00	1.61E+00
	18	**5.06E+00**	5.64E+00	**4.81E+00**	5.51E+00	**4.38E+00**
	19	3.44E+00	3.40E+00	3.56E+00	**3.34E+00**	3.57E+00

Para $k = 1$, R-PSO se comporta como un algoritmo de búsqueda local que no acepta malas soluciones. Esto se debe a que si una partícula, guiada por la experiencia personal de sus vecinas, realiza un movimiento que la ubica en una posición con peor evaluación que la que tenía antes, entonces, debido al retorno, la nueva posición es desechada. En otras palabras, una partícula acepta una nueva posición solo si esta es mejor que la anterior. Este comportamiento reduce a cero las posibilidades, ya limitadas de PSO

Tabla 3.2: Errores absolutos de R-PSO$_l$ y RE-PSO$_l$

Algoritmo	fn	$k=1$	$k=10$	$k=50$	$k=100$	$k=1000$
R-PSO$_l$	15	2.91E+02	**5.13E+01**	**5.07E+01**	**5.12E+01**	**5.02E+01**
	16	1.87E+01	6.36E+00	**6.21E+00**	6.45E+00	**6.01E+00**
	17	5.74E+00	6.19E-01	6.04E-01	6.13E-01	7.47E-01
	18	2.03E+01	**2.97E+00**	**2.76E+00**	3.11E+00	3.29E+00
	19	9.51E+00	**3.12E+00**	3.35E+00	**3.25E+00**	3.40E+00
RE-PSO$_l$	15	**5.15E+01**	**5.22E+01**	**5.13E+01**	**5.20E+01**	5.61E+01
	16	**6.25E+00**	6.65E+00	6.43E+00	6.33E+00	6.47E+00
	17	6.33E-01	6.41E-01	7.07E-01	6.52E-01	6.34E-01
	18	**2.85E+00**	**2.99E+00**	3.33E+00	**2.91E+00**	3.11E+00
	19	**3.26E+00**	3.35E+00	3.34E+00	**3.30E+00**	**3.30E+00**

Tabla 3.3: Diferencias significativas entre R-PSO$_g$, RE-PSO$_g$ y PSO$_g$

Algoritmo	fn	$k=1$	$k=10$	$k=50$	$k=100$	$k=1000$
R-PSO$_g$	15	**0.0%**	18.3%	98.2%	76.2%	39.8%
	16	**0.0%**	7.2%	23.2%	19.5%	**1.3%**
	17	**0.0%**	9.5%	41.9%	89.6%	65.8%
	18	**0.0%**	25.8%	31.2%	51.0%	96.1%
	19	**0.0%**	28.1%	61.4%	33.2%	56.9%
RE-PSO$_g$	15	80.9%	40.2%	77.8%	38.9%	99.7%
	16	92.7%	66.0%	56.3%	34.0%	8.5%
	17	11.1%	12.4%	22.5%	**2.3%**	20.6%
	18	90.2%	49.4%	58.9%	60.3%	24.1%
	19	72.7%	90.8%	29.6%	86.1%	28.6%

estándar, de escapar de un óptimo local. En este sentido, es de esperar que sus resultados sean peores a los del estándar.

En RE-PSO, mientras tanto, el retorno de una partícula hacia la mejor posición lograda en las últimas k iteraciones (*pkbest*) solo se efectúa si esta es mejor que la encontrada en iteraciones pasadas, en caso contrario, la partícula continua su curso natural. Para $k=1$, se tiene que *pkbest* es igual a la posición actual por lo que, se efectúe o no el retorno, la partícula no cambia su posición actual. Esto implica que el comportamiento del algoritmo es semejante al del estándar, de modo que sus resultados deberían ser similares.

En general, el retorno convierte al PSO estándar, ya sea global o local, en un algoritmo más de explotación y menos exploratorio. Esto se debe a que la intensificación de la búsqueda alrededor de las mejores soluciones encontradas en previas iteraciones reduce sus capacidades de exploración, pues restringe la búsqueda a regiones ya exploradas. Este efecto en etapas iniciales, cuando aún no se ha explorado lo suficiente, puede traer consecuencias negativas, por ejemplo, una convergencia prematura, pero si se aplica

Tabla 3.4: Diferencias significativas entre R-PSO$_l$, RE-PSO$_l$ y PSO$_l$

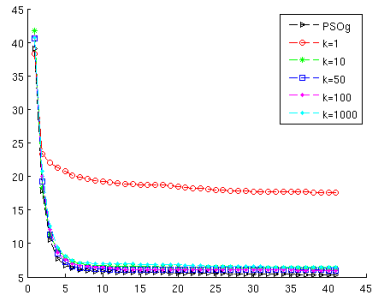
Algoritmo	fn	$k=1$	$k=10$	$k=50$	$k=100$	$k=1000$
R-PSO$_l$	15	**0.0%**	57.1%	38.6%	54.1%	34.5%
	16	**0.0%**	91.7%	72.1%	70.5%	35.4%
	17	**0.0%**	43.2%	57.6%	43.0%	**1.3%**
	18	**0.0%**	56.3%	13.7%	93.2%	37.5%
	19	**0.0%**	**4.8%**	76.2%	56.6%	42.1%
RE-PSO$_l$	15	61.1%	81.3%	58.9%	78.6%	23.0%
	16	79.6%	33.5%	72.0%	99.5%	67.7%
	17	35.3%	18.5%	**2.9%**	18.8%	19.9%
	18	26.8%	66.6%	33.1%	31.7%	93.0%
	19	60.3%	69.6%	80.6%	89.6%	89.4%

tras identificar las regiones más prometedoras del espacio, existen más posibilidades de que la convergencia sea en la solución óptima. Dado que la capacidad de exploración del modelo local es mayor que la del modelo global, los efectos negativos del retorno son más notables en la variante global. Esto puede apreciarse en las tablas 3.1 y 3.2, donde la variante local con retorno obtiene mejoras en 21 casos, mientras que la global solo en 11.

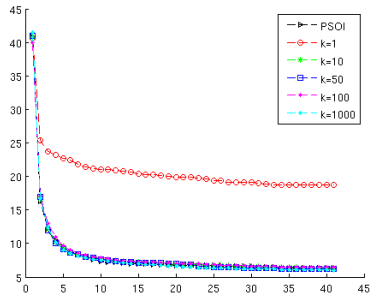
No obstante a las mejoras observadas respecto al error absoluto, puede apreciarse en las tablas 3.3 y 3.4 que apenas existe diferencia entre los resultados obtenidos por las implementaciones con retorno y por las del estándar. Solo para $k = 1$ y en 2 o 3 casos más las diferencias entre ambos algoritmos son significativas y solo en uno de estos casos la variante con retorno supera a la estándar (R-PSO$_l$ con $k = 10$ en F19). Probablemente esto se deba a que una vez que el enjambre ha convergido a una región del espacio, el comportamiento de un PSO con retorno sea muy similar al de uno estándar.

Analizando las reglas que rigen el funcionamiento de ambos algoritmos, se puede apreciar que, durante la mayor parte de su ejecución, R-PSO funciona como un PSO estándar. La diferencia tiene lugar en marcadas iteraciones al producirse el retorno mediante el cual todas las partículas se ven atraídas de manera súbita por sus mejores posiciones.

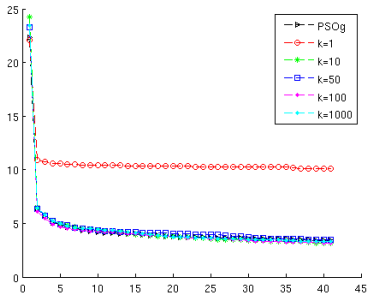
A primera vista debiera implicar una gran diferencia que, por un lado, las partículas sean atraídas por su *pbest* y *gbest* gradualmente y, por otro, que sean ubicadas precipitadamente en su *pbest*. En realidad, el comportamiento del enjambre bajo un procedimiento u otro es diferente en dependencia de la ubicación de cada partícula respecto a *pbest* y *gbest*. Si una partícula se encuentra muy cerca de estas posiciones, entonces la diferencia entre su posición actual y la que ocupe luego de efectuarse el retorno será apenas imperceptible. En caso contrario, si la partícula se encuentra muy alejada de estas posiciones parecería como si diera un salto en el espacio de soluciones.



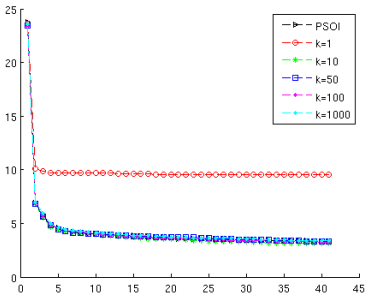
(a) Versión global en F16



(b) Versión local en F16



(c) Versión global en F19

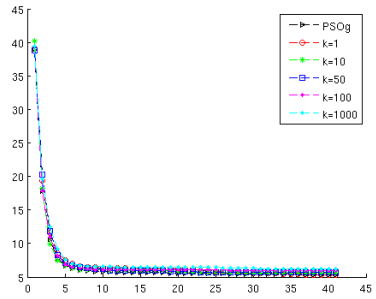


(d) Versión local en F19

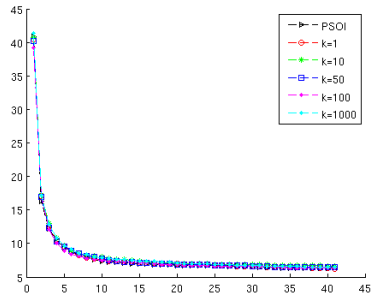
Figura 3.1: Convergencia de PSO estándar y R-PSO para diferentes valores de k en F16 y F19. Los gráficos se contruyeron con el promedio del error de la mejor solución encontrada cada 50 iteraciones en las 50 corridas realizadas por cada algoritmo.

Suponiendo que la composición inicial del enjambre sea lo suficientemente diversa, el efecto del retorno se hace más notable al comienzo de la ejecución del algoritmo, en cambio, a medida que el enjambre va convergiendo este se hace menos perceptible. Con la convergencia las partículas se van conglomerando en una misma región del espacio, sus posiciones tienden a ser muy similares y a estar más cerca de su *pbest* y *gbest*. De modo que, con la convergencia del enjambre el comportamiento de PSO con retorno se va haciendo similar al de PSO estándar.

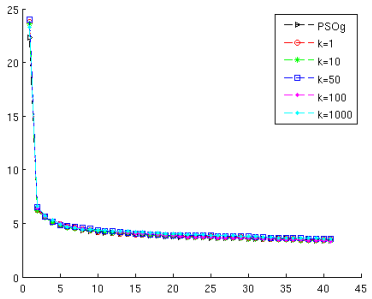
Estas observaciones también pueden apreciarse en los gráficos de convergencia que se muestran en las figuras 3.1 y 3.2, correspondientes a la evolución de la mejor solución encontrada por R-PSO y RE-PSO para las



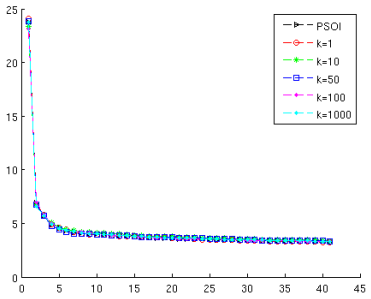
(a) Versión global en F16



(b) Versión local en F16



(c) Versión global en F19



(d) Versión local en F19

Figura 3.2: Convergencia de PSO estándar y RE-PSO para diferentes valores de k en F16 y F19. Los gráficos se contruyeron con el promedio del error de la mejor solución encontrada cada 50 iteraciones en las 50 corridas realizadas por cada algoritmo.

funciones F16 y F19. Los gráficos de convergencia para el resto de las funciones se encuentran en los anexos del documento (3.4.5).

Estos gráficos fueron creados tomando la evaluación de la mejor solución encontrada por todo el enjambre cada 50 iteraciones y calculando el error respecto al valor óptimo de cada función. En cada gráfico puede observarse el promedio de estos errores para cada uno de los períodos de 50 iteraciones completados.

Como se puede apreciar, el comportamiento de las implementaciones con retorno y retorno con exploración comienzan a ser muy similares a las del estándar desde iteraciones bien tempranas para todos los valores de k, excepto para R-PSO con $k = 1$, lo cual confirma la primera conclusión.

Considerando las características de las funciones utilizadas en los experimentos, se concluye además, que la estrategia de retorno no resulta efectiva para resolver funciones con múltiples óptimos. De acuerdo a su funcionamiento, el retorno hace de PSO un algoritmo menos exploratorio y más sensible a quedar atrapado en un óptimo local, contrario a lo que sería un mecanismo eficiente de exploración, lo cual es un requisito indispensable para efectuar la búsqueda de la solución óptima en este tipo de funciones.

3.4.5. Análisis de PSO con Actualización Retardada

En las tablas 3.5 y 3.6 se muestran los errores absolutos obtenidos por los algoritmos PSO_g y PSO_l con la estrategia de Actualización Retardada y por su combinación con las estrategias de Retorno y Retorno con Exploración como se describió en la sección 3.3. Los resultados de las pruebas realizadas para precisar si las diferencias entre estos algoritmos y las variantes del estándar son significativas, se pueden encontrar en las tablas 3.7 y 3.8.

Después de analizar y comparar estos resultados se puede llegar a las siguientes conclusiones:

- Para $k = 1$ la actualización retardada no introduce variación en el comportamiento de ninguno de los algoritmos.
- Para $k = 1000$ la actualización retardada deteriora significativamente el rendimiento de PSO estándar.
- PSO con actualización retardada produce mejores resultados que su combinación con cualquiera de las dos formas de retorno.
- La incomunicación temporal entre las partículas, producto de la actualización retardada, beneficia a ambos modelos de una manera significativa respecto al estándar.

De acuerdo a los resultados de las columnas correspondientes a $k = 1$ en cada una de las tablas, se observa que, excepto para RAR-PSO, el rendimiento de los algoritmos es similar al del estándar. Esto se debe a que para $k = 1$ la actualización de *pbest* y *gbest* ocurre en cada iteración, y por tanto, no se introduce variación en los algoritmos, de ahí que, el comportamiento de AR-PSO, RAR-PSO y REAR-PSO sea igual al de PSO, R-PSO y RE-PSO, respectivamente. En el análisis de los efectos de la estrategia de retorno en la sección anterior, se determinó que, para este valor de k, RE-PSO se comporta como un PSO estándar, mientras que, R-PSO como un algoritmo de búsqueda local sin mecanismo para escapar de óptimos locales. Esto explica,

entonces, la similitud observada entre los resultados de AR-PSO, REAR-PSO y la versión estándar de PSO, así como el deterioro en el rendimiento de la variante RAR-PSO.

Tabla 3.5: Errores absolutos de AR-PSO$_g$, RAR-PSO$_g$ y REAR-PSO$_g$

Algoritmo	fn	$k=1$	$k=10$	$k=50$	$k=100$	$k=1000$
AR-PSO$_g$	15	**6.20E+01**	**5.21E+01**	**5.81E+01**	**6.17E+01**	2.13E+02
	16	6.05E+00	**4.62E+00**	**3.95E+00**	**4.34E+00**	1.68E+01
	17	1.65E+00	**5.13E-01**	**7.96E-01**	**6.18E-01**	4.04E+00
	18	5.33E+00	**2.75E+00**	**2.73E+00**	**2.65E+00**	1.46E+01
	19	3.49E+00	3.41E+00	3.57E+00	3.71E+00	5.70E+00
RAR-PSO$_g$	15	2.73E+02	**6.25E+01**	**6.10E+01**	**6.11E+01**	2.12E+02
	16	1.69E+01	**5.29E+00**	**3.76E+00**	**4.38E+00**	1.65E+01
	17	6.47E+00	**8.90E-01**	**8.15E-01**	**5.88E-01**	4.07E+00
	18	2.15E+01	**3.45E+00**	**2.43E+00**	**2.57E+00**	1.52E+01
	19	1.00E+01	3.41E+00	3.78E+00	3.95E+00	5.77E+00
REAR-PSO$_g$	15	6.38E+01	6.28E+01	6.35E+01	6.53E+01	2.15E+02
	16	5.62E+00	5.69E+00	**3.67E+00**	**3.98E+00**	1.62E+01
	17	1.39E+00	**8.52E-01**	**6.34E-01**	**5.71E-01**	4.02E+00
	18	5.50E+00	**3.65E+00**	**2.77E+00**	**2.46E+00**	1.50E+01
	19	**3.13E+00**	3.55E+00	3.77E+00	3.93E+00	5.62E+00

Tabla 3.6: Errores absolutos de AR-PSO$_l$, RAR-PSO$_l$ y REAR-PSO$_l$

Algoritmo	fn	$k=1$	$k=10$	$k=50$	$k=100$	$k=1000$
AR-PSO$_l$	15	**4.90E+01**	**4.64E+01**	5.97E+01	7.93E+01	2.75E+02
	16	6.49E+00	**5.57E+00**	**5.59E+00**	**5.90E+00**	1.57E+01
	17	7.14E-01	**4.05E-01**	**4.01E-01**	7.46E-01	5.65E+00
	18	**3.02E+00**	**1.82E+00**	**2.36E+00**	3.11E+00	2.05E+01
	19	**3.24E+00**	**3.01E+00**	3.42E+00	3.52E+00	6.64E+00
RAR-PSO$_l$	15	2.89E+02	6.03E+01	6.28E+01	8.08E+01	2.77E+02
	16	1.84E+01	**5.73E+00**	**5.62E+00**	**5.62E+00**	1.58E+01
	17	5.76E+00	**4.60E-01**	**4.82E-01**	6.87E-01	5.49E+00
	18	2.02E+01	**2.36E+00**	**2.19E+00**	**3.02E+00**	2.03E+01
	19	9.74E+00	**3.08E+00**	3.67E+00	3.69E+00	6.84E+00
REAR-PSO$_l$	15	**5.02E+01**	5.81E+01	6.70E+01	8.24E+01	2.83E+02
	16	6.65E+00	**5.89E+00**	**5.90E+00**	6.70E+00	1.55E+01
	17	6.20E-01	**4.91E-01**	**4.72E-01**	7.88E-01	5.56E+00
	18	3.18E+00	**2.29E+00**	**2.35E+00**	**3.06E+00**	2.08E+01
	19	**3.30E+00**	3.37E+00	3.73E+00	3.81E+00	6.67E+00

Como ya se conoce, el proceso de optimización en PSO tiene lugar mediante el sucesivo movimiento de las partículas de una posición a otra en el espacio, guiadas por las posiciones *pbest* y *gbest*, que representan las mejores soluciones encontradas hasta el momento. En este proceso, un factor determinante, y que caracteriza el funcionamiento de este paradigma

de optimización, es el intercambio de información que se produce entre las partículas mediante la actualización de estas posiciones. Sin estas actualizaciones, las partículas son agentes recorriendo el espacio de búsqueda de manera independiente y guiadas, principalmente, por factores estocásticos. Esto es precisamente lo que sucede con AR-PSO para $k = 1000$, donde el intercambio de información solo tiene lugar una vez, luego de una cantidad considerable de iteraciones. Durante dos largos períodos de incomunicación, las partículas realizan una excesiva exploración alrededor de las posiciones *pbest* y *gbest* que no necesariamente se mantendrán como las mejores soluciones encontradas. De esta forma, al considerar regiones bien limitadas para explorar, las posibilidades de encontrar la solución óptima, incluso una solución óptima local, son muy reducidas.

Como se puede apreciar, en las tablas 3.5 y 3.6, los resultados de esta variante para $k = 1000$ son inferiores y significativamente diferentes (tablas 3.7 y 3.8) a los del estándar para todas las funciones. En el gráfico de convergencia (Figura 3.3) se observa claramente la mejoría que se produce en las soluciones una vez que las partículas intercambian información en el período 20, al completarse las 1000 iteraciones, confirmando la importancia del intercambio de experiencias para el proceso de optimización del algoritmo. Se aprecia además, que hasta el momento del intercambio, y después de realizado este, la convergencia del algoritmo transcurre lentamente.

Tabla 3.7: Diferencias significativas entre AR-PSO$_g$, RAR-PSO$_g$, REAR-PSO$_g$ y PSO$_g$

Algoritmo	fn	$k = 1$	$k = 10$	$k = 50$	$k = 100$	$k = 1000$
AR-PSO$_g$	15	88.7%	**0.8%**	38.6%	85.9%	**0.0%**
	16	14.9%	8.6%	**0.0%**	**1.3%**	**0.0%**
	17	8.6%	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	18	78.5%	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	19	58.3%	84.0%	20.8%	**5.0%**	**0.0%**
RAR-PSO$_g$	15	**0.0%**	98.1%	74.1%	74.2%	**0.0%**
	16	**0.0%**	84.7%	**0.0%**	**3.5%**	**0.0%**
	17	**0.0%**	**0.1%**	**0.1%**	**0.0%**	**0.0%**
	18	**0.0%**	**0.2%**	**0.0%**	**0.0%**	**0.0%**
	19	**0.0%**	84.7%	**0.9%**	**0.0%**	**0.0%**
REAR-PSO$_g$	15	81.4%	96.5%	84.2%	55.7%	**0.0%**
	16	58.6%	40.9%	**0.1%**	**0.5%**	**0.0%**
	17	99.0%	**0.0%**	**0.0%**	**0.0%**	**0.0%**
	18	63.7%	**0.6%**	**0.0%**	**0.0%**	**0.0%**
	19	16.1%	28.8%	**2.7%**	**0.0%**	**0.0%**

Con respecto a las variantes con retorno, se puede verificar que en general ambos modelos se favorecen un poco más de la estrategia de actualización

Tabla 3.8: Diferencias significativas entre AR-PSO$_l$, RAR-PSO$_l$, REAR-PSO$_l$ y PSO$_l$

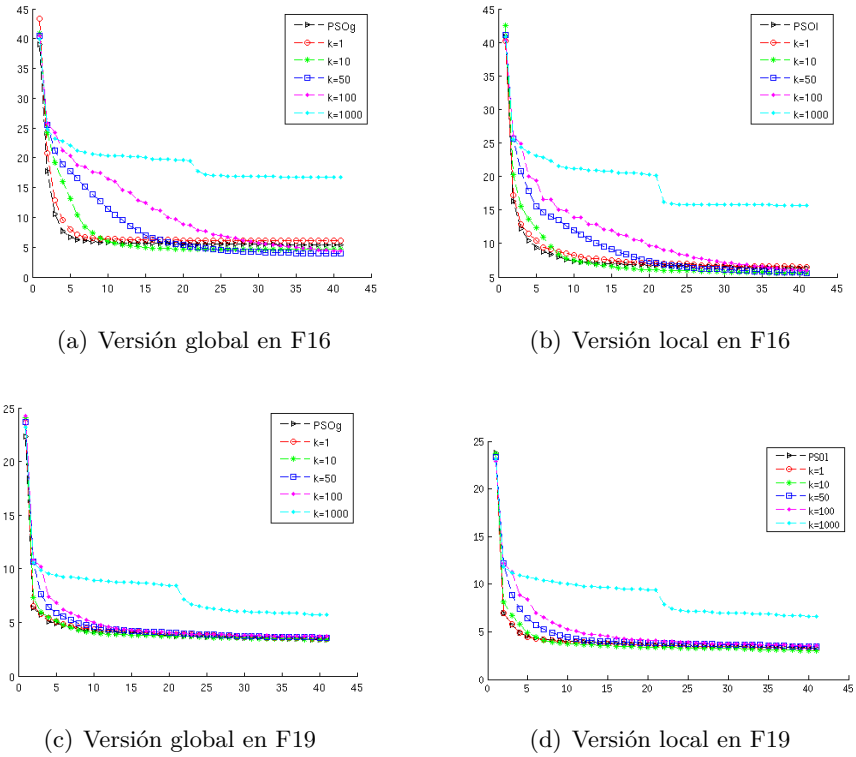
Algoritmo	fn	$k=1$	$k=10$	$k=50$	$k=100$	$k=1000$
AR-PSO$_l$	15	19.4%	**2.7%**	**2.1%**	**0.0%**	**0.0%**
	16	62.2%	**1.0%**	**1.1%**	18.7%	**0.0%**
	17	6.5%	**0.4%**	**0.1%**	**0.2%**	**0.0%**
	18	76.3%	**0.0%**	**0.1%**	91.3%	**0.0%**
	19	47.3%	**1.1%**	26.5%	**2.5%**	**0.0%**
RAR-PSO$_l$	15	**0.0%**	**0.3%**	**0.3%**	**0.0%**	**0.0%**
	16	**0.0%**	**3.0%**	**1.9%**	**1.0%**	**0.0%**
	17	**0.0%**	**4.5%**	9.0%	**3.2%**	**0.0%**
	18	**0.0%**	**0.0%**	**0.0%**	68.0%	**0.0%**
	19	**0.0%**	**4.2%**	**0.1%**	**0.0%**	**0.0%**
REAR-PSO$_l$	15	26.6%	7.9%	**0.0%**	**0.0%**	**0.0%**
	16	32.9%	11.6%	21.6%	21.8%	**0.0%**
	17	48.6%	15.2%	**3.3%**	**0.0%**	**0.0%**
	18	67.9%	**0.0%**	**0.0%**	89.6%	**0.0%**
	19	89.7%	53.4%	**0.0%**	**0.0%**	**0.0%**

retardada que de su combinación con cualquiera de las dos formas de retorno. Esto se debe a que el retorno reduce la capacidad de exploración del algoritmo, lo cual no es favorable para el tipo de funciones utilizadas en los experimentos.

Otro resultado para destacar es que, para ambos modelos de vecindades, la actualización de las posiciones *pbest* y *gbest* cada $k = 10, 50, 100$ iteraciones parece favorecer el proceso de búsqueda de PSO. Como se puede observar en las tablas 3.7 y 3.8, a diferencia de la reducción de la dimensión y del retorno, con la actualización retardada se logra modificar significativamente el comportamiento de PSO en favor de las funciones multimodales.

En el capítulo 3 se plantearon algunas ideas de cómo obtener un modelo de vecindad con la capacidad de exploración del modelo local, y al mismo tiempo, con la rapidez para converger del modelo global. Teniendo en cuenta que la capacidad de exploración en el modelo local está muy asociada con la comunicación parcial que se produce entre las partículas, se puede considerar a la actualización retardada – comunicación parcial entre las partículas debido a su aislamiento durante k iteraciones – como una forma de aumentar la capacidad de exploración del modelo global.

La comunicación parcial en el modelo local se debe a la manera en que están organizadas las partículas en subenjambres, con solapamientos, de modo que, al menos de una manera indirecta, todas las partículas están comunicadas entre sí. Esta forma de agrupar las partículas hace que la localización de un óptimo local sea difundido por todo el enjambre de una manera más



(a) Versión global en F16

(b) Versión local en F16

(c) Versión global en F19

(d) Versión local en F19

Figura 3.3: Convergencia de PSO estándar y AR-PSO para diferentes valores de k en F16 y F19. Los gráficos se contruyeron con el promedio del error de la mejor solución encontrada cada 50 iteraciones en las 50 corridas realizadas por cada algoritmo.

lenta que como se haría en el modelo global; de ahí que las posibilidades de explorar otras soluciones y encontrar otros óptimos sea mucho mayor en el primero que en el segundo.

La comunicación parcial introducida por la estrategia de actualización retardada en el modelo global, no es igual pero tiene un efecto similar en este sentido. Se podría decir que se trata de una incomunicación temporal, que restringe el intercambio de experiencias entre las partículas, produciéndose este al final de cada período de exploración. Como resultado de este aislamiento, si una partícula es atraída por un óptimo local, pasará algún tiempo hasta que pueda hacer público su éxito y comience a llevarse a todo el enjambre consigo. Al menos durante varias iteraciones las demás partícu-

las seguirán explorando en busca de mejores óptimos, y como consecuencia, la probabilidad de escapar del óptimo local encontrado será mayor.

Este comportamiento es más probable al comienzo de la ejecución del algoritmo, cuando la diversidad de soluciones es mayor y las partículas exploran diferentes regiones del espacio de búsqueda de acuerdo a sus posiciones iniciales. Una vez que todo el enjambre comienza a converger a una solución óptima, esto implica una disminución de la diversidad en el enjambre, y por tanto, la probabilidad de que se encuentre otra solución óptima es menor. En este estado, el efecto de la actualización retardada es diferente y pudiera describirse como un mecanismo para explotar las soluciones que rodean las mejores soluciones encontradas hasta el momento. A partir de este comportamiento, podría entenderse entonces el beneficio que produce la estrategia al modelo local. Uno de los señalamientos que se hace a este modelo es precisamente la lentitud para converger producto de su capacidad para explorar diferentes óptimos. Al restringir el intercambio de información durante varias iteraciones, se intensifica la búsqueda en las regiones exploradas favoreciendo la mejoría de las mejores soluciones encontradas, y al mismo tiempo, se reduce la exploración de nuevas soluciones.

De esta forma, puede verse la estrategia de actualización retardada como un mecanismo de dos fases. Incialmente, favorece la exploración y reduce la probabilidad de convergencia prematura del enjambre a un óptimo local, y luego, a medida que se produce la convergencia del enjambre y disminuye la diversidad de soluciones, su comportamiento tiende a favorecer la explotación de las regiones exploradas. Este mecanismo resulta conveniente cuando se quiere resolver una función multimodal, pues brinda una técnica de exploración que permite evadir los óptimos locales en etapas iniciales de la búsqueda. Los resultados obtenidos, por ejemplo, para $k = 10, 50$ muestran cómo ambos modelos de vencindades se favorecen de esta técnica, reportando mejores resultados que la variante estándar en este tipo de funciones.

Aplicaciones

Además de favorecer el proceso de búsqueda de PSO en favor de las funciones con múltiples óptimos, la estrategia de actualización retardada puede ser utilizada para mejorar las implementaciones paralelas de PSO. A continuación se describe brevemente en que se basa esta conclusión.

Los algoritmos descritos en este trabajo han sido implementados en serie, es decir, cada una de las intrucciones que los componen son ejecutadas de manera secuencial. El costo o la complejidad de los algoritmos secuenciales se estima en términos del espacio en memoria y el tiempo de ejecución que

puede tomar la realización de todas sus instrucciones.

Debido a la gran cantidad de evaluaciones de función y actualizaciones que deben realizar a los miembros de la población en cada iteración, las metaheurísticas basadas en poblaciones, como PSO, suelen ser mucho más costosas que los algoritmos basados en trayectorias y los métodos exáctos. Estos han demostrado ser bastante eficientes en la solución de problemas de pequeñas dimensiones, sin embargo, en muchas ocasiones su aplicación se vuelve impracticable cuando se trata de problemas de grandes dimensiones o problemas que requieren de soluciones en tiempo real. Para poder resolver este tipo de problemas, la implementación de algoritmos paralelos se ha convertido, prácticamente, en una necesidad.

La computación paralela es una técnica de programación en la que muchas instrucciones se ejecutan simultáneamente. Se basa en el principio de que los problemas grandes se pueden dividir en partes más pequeñas que pueden resolverse de forma concurrente o en paralelo [52]. En los últimos años, la implementación de algoritmos palalelos ha cobrado gran auge debido a la existencia de computadoras con múltiples procesadores, clústeres, sistemas distribuidos y la posibilidad más reciente de utilizar las unidades funcionales de las GPU (Unidades de Procesamiento Gráfico [53]) para realizar operaciones en paralelo. Los algoritmos cuyas instrucciones son realizadas concurrentemente tienen como ventaja que pueden reducir su tiempo de ejecución, mejorar las soluciones obtenidas y solucionar problemas que por su complejidad requieren de procesadores más potentes para realizar sus cálculos.

Además del espacio en memoria y el tiempo de ejecución, el costo de los algoritmos paralelos se mide en términos de la comunicación que se produce entre los diferentes procesadores, computadoras o unidades funcionales, que puede realizarse a través del intercambio de mensajes y/o del acceso a memoria compartida. En este sentido, los algoritmos paralelos poseen un recurso más a tener en cuenta (optimizar) con respecto a los secuenciales.

Actualmente existe una vasta literatura referente a la programación paralela, así como de metodologías para la paralelización de la mayoría de las metaheurísticas (ver [53, 54]). Entre los modelos más difundidos para metaheurísticas basadas en poblaciones, se encuentran los modelos cooperativos a nivel de algoritmo (e.g. Modelo de Islas) y los modelos a nivel de iteraciones (e.g. Modelo Maestro-Esclavo).

En la mayoría de las referencias consultadas sobre PSO paralelos se describe el empleo del Modelo Maestro-Esclavo (Master-Slave Model). En este modelo se hace énfasis en la paralelización de las instrucciones que se realizan en cada una de las iteraciones. Usualmente, aquellas operaciones que se

llevan a cabo sobre cada partícula del enjambre son las que se ejecutan de manera concurrente (e.g. evaluación de la función objetivo, actualización de las velocidades y posiciones, etc.).

Las primeras implementaciones de este modelo (e.g. [54]) se conocen como síncronas, pues se basaron en la versión original de PSO, donde la actualización del *gbest* espera porque se realice la evaluación y la actualización del *pbest* de todas las partículas. Las variantes asíncronas (e.g. [55, 56]) surgieron luego como alternativa para disminuir el tiempo de ejecución del algoritmo en sistemas distribuídos y ambientes paralelos compuestos por procesadores heterogéneos, debido a que el costo final del algoritmo era dependiente de la capacidad de procesamiento del procesador menos potente. La explicación, en cierto sentido lógica, es que en las variantes síncronas las iteraciones están sincronizadas y, por tanto, se debe esperar por el resultado de todas las operaciones enviadas a realizar sobre cada una de las partículas, haciendo que el tiempo de ejecución del procesador más lento sea el determinante. En las variantes asíncronas la actualización del *gbest* se realiza inmediatamente tras la evaluación de cada partícula (similar a un PSO con actualización asíncrona), causando una atracción instantánea hacia las nuevas mejores soluciones encontradas, y por tanto, un aumento de la posibilidad de una convergencia prematura. Excepto en [38], en todas las referencias consultadas se manifiesta que los resultados (solución final) obtenidos por la variante asíncrona es superior a los de la variante síncrona.

Del resto de las referencias consultadas resalta la reciente publicación [57] que propone una implementación asíncrona del Modelo Celular (caso particular del Modelo de Islas donde cada isla se compone de un solo miembro), en la que las partículas evolucionan de manera independiente, ejecutando en paralelo (un hilo de ejecución por dimensión) la evaluación de su posición y la actualización de su velocidad, posición actual y *pbest*. Como estructura de comunicación utilizan una topología de anillo con $n = 2$. Según sus autores, esta nueva variante, implementada usando las unidades funcionales de los GPU modernos, supera, tanto en tiempo de ejecución como en calidad de soluciones, a implementaciones síncronas y asíncronas usando el modelo Maestro-Esclavo.

En resumen, la paralelización de PSO parece ser la tendencia actual para reducir su alto costo computacional. En este sentido, aunque resulta esencial escoger un ambiente paralelo que permita reducir al máximo el tiempo de ejecución, el diseño del algoritmo es también muy importante. Por un lado, las implementaciones paralelas asíncronas parecen estar imponiéndose a las síncronas, debido a que logran sobreponerse a la heterogeneidad de los sistemas paralelos existentes. Por otro, se tiene la selección adecuada de

la estructura topológica para el intercambio de información y comunicación entre procesadores. A diferencia de otras metaheurísticas, como los Algoritmos Genéticos, donde la evolución es poblacional (mediante operadores que involucran a más de un individuo como el de cruzamiento), en PSO las partículas evolucionan independientemente, pero sobre la base de un gran intercambio de información. Es decir, que el intercambio de información en un PSO paralelo no es consecuencia propiamente de la paralelización del algoritmo sino un componente intrínseco en su definición y necesario para su buen desempeño. En este sentido, se podría decir que implementaciones paralelas de una versión global de PSO (PSO_g) generan, con respecto a una versión local de PSO (PSO_l), un exceso de comunicación entre los procesadores dado a que todas las partículas necesitan intercambiar información con todas las demás. Debido a esto, y a que, de los dos modelos, el local tiende a reportar mejores resultados, parece lógico que las implementaciones paralelas de PSO actuales (e.g. [57]) consideren la utilización de una topología local en lugar de una global.

Considerando los resultados obtenidos por las dos implementaciones del algoritmo AR-PSO, se prevee que el uso de la estrategia de Actualización Retardada podría resultar favorable para las implementaciones paralelas de PSO. Por un lado, disminuyendo el tiempo de ejecución del algoritmo, debido a la incomunicación temporal que produce una reducción en el intercambio de información entre las partículas/los procesadores, y por otro, aumentando la calidad de las soluciones obtenidas.

Conclusiones

Durante el desarrollo de este trabajo se realizó una amplia revisión bibliográfica de la metaheurística Optimización de Enjambres de Partículas; se estudiaron sus antecedentes, los componentes básicos de la versión original, las diferentes modificaciones realizadas desde su introducción como técnica de búsqueda y optimización global, así como la nueva versión estándar del algoritmo.

Dada la ausencia explícita de estrategias para escapar de óptimos locales, en la definición estándar de PSO, y la identificación de esta dificultad por muchos investigadores, como un problema de balance entre exploración y explotación, se experimentaron con diferentes estrategias de búsqueda que sirvieran para mejorar el balance entre estas dos importantes componentes. El diseño de las nuevas técnicas tuvo como influencia algunas de las estrategias de búsqueda propuestas por otros investigadores para el algoritmo, así como para otras metaheurísticas.

De la estrategia Reducción de la Dimensión, que introduce el algoritmo Locust Swarms, surgió el algoritmo MPSO, que modifica la trayectoria de las partículas al reducir el número de dimensiones que son exploradas durante el proceso de búsqueda de PSO. Se experimentó con tres variantes del algoritmo, que difieren en el mecanismo de selección de las dimensiones a explorar, y se evaluó el desempeño de cada una con dos conjuntos de funciones con características separables y multimodales. De acuerdo a los resultados obtenidos, se concluyó que la estrategia mejora el rendimiento de PSO estándar en funciones separables. Producto de la comparación entre las diferentes variantes se determinó que para obtener buenos resultados, el mecanismo de selección de las dimensiones que se utilice debe asegurar que todas las dimensiones sean exploradas y, además, que el beneficio es mayor si la selección no es aleatoria.

Tomando como motivación la fase de intensificación que realizan algunas metaheurísticas, como Búsqueda Tabú, surgió la estrategia de Retorno. El

análisis de las dos formas implementadas para ejecutar el retorno revelaron que el comportamiento de los algoritmos tendía a ser similar al de PSO estándar a medida que se producía la convergencia del enjambre. Los resultados con estas variantes mostraron que la estrategia no es efectiva para resolver funciones con múltiples óptimos, debido al efecto de reducción de la capacidad de exploración que produce en el algoritmo en etapas iniciales de su ejecución.

La Actualización Retardada se introdujo como una estrategia para prolongar la fase de exploración del algoritmo, mediante la incomunicación temporal de las partículas. Los resultados obtenidos confirman que la estrategia aumenta la capacidad de exploración de PSO y favorece la búsqueda de mejores soluciones para funciones multimodales.

De las tres estrategias propuestas en este trabajo, se puede considerar a la Reducción de la Dimensión y a la Actualización Retardada, como las de resultados más relevantes y con posibilidades de aplicación. La primera constituye una forma de capacitar el PSO para explotar la separabilidad de una función. El análisis de las variantes implementadas puede servir de motivación para la experimentación con nuevos mecanismos de selección, que permitan obtener resultados como los obtenidos por la variante con búsqueda exhaustiva o mejores, a un costo menor.

La computación paralela constituye actualmente una de las vías más explotadas para reducir el costo computacional de las metaheurísticas basadas en poblaciones. Uno de los aspectos que más se discute en las implementaciones paralelas que se realizan de PSO es cómo reducir la sobrecarga de comunicación que se produce entre los procesadores y así el costo computacional del algoritmo. Los resultados obtenidos con la estrategia de Actualización Retardada sugieren que su uso podría favorecer, significativamente, las implementaciones paralelas de PSO respecto al tiempo de ejecución, así como a la calidad de sus soluciones.

Se puede concluir que los objetivos planteados al comienzo del presente trabajo de investigación fueron cumplidos satisfactoriamente. Los resultados alcanzados revelan técnicas útiles que pueden ser utilizadas en futuras variantes continuas o discretas de PSO, en algoritmos basados en su funcionamiento, así como en hibridaciones con otras metaheurísticas; y crea un precedente para el desarrollo de futuros proyectos en el área de la Optimización de Enjambres de Partículas Paralelas (PPSO).

Como un dato que avala la novedad de esta investigación, los resultados presentados en este capítulo fueron expuestos oralmente en la 9^{na} *Edición de la Conferencia Internacional Mexicana en Inteligencia Artificial (MICAI 2010)*, y publicados en los *proceedings* de la conferencia [58].

Recomendaciones

Esta investigación, lejos de recién comenzar, constituye una continuación de la que comenzaron James Kennedy y Russel Eberhart, creadores de PSO, en el año 1995. Desde la introducción de PSO en el mundo de la optimización, su perfeccionamiento, así como el de cualquier metaheurística, es un campo de investigación que cada día se hace más amplio e interesante. Dadas las posibilidades de mejorar el balance entre exploración y explotación de una metaheurística, utilizando las técnicas probadas en otras, así como con la estrategia de la hibridación, se podrían enumerar muchísimas maneras de intentar mejorar el desempeño de PSO.

Como continuación de la prueba y validez de los resultados alcanzados en este trabajo, se recomienda continuar perfeccionado las estrategias y diseñar otras que mejoren el balance entre exploración y explotación de PSO:

- Los efectos de las estrategias propuestas han sido analizados sobre implementaciones de PSO con actualización síncrona. Sería interesante evaluar qué efectos producen en implementaciones de PSO con actualización asíncrona.

- Diseñar nuevos mecanismos de selección de las dimensiones para mejorar los resultados de MPSO, en cuanto a la calidad de las soluciones y al número de evaluaciones de función que se necesitan para obtenerlas.

- Medir la escalabilidad de los algoritmos propuestos evaluándolos en problemas de dimensiones más grandes y su desempeño en problemas con aplicaciones reales.

- Comprobar la factibilidad de extender las estrategias propuestas a implementaciones de PSO binarias y discretas.

- Implementar la variante de PSO con Actualización Retardada (ARPSO) en paralelo y comparar, con otras implementaciones paralelas

de PSO, su desempeño en cuanto a tiempo de ejecución y calidad de las soluciones.

Bibliografía

[1] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 5:533–549, 1986. (Citado en la página 1).

[2] P. Hansen and N. Mladenović. *An Introduction to variable neighborhood search.*, chapter 30, pages 433–458. Kluwer Academic Publishers, 1999. (Citado en la página 1).

[3] T. Stützle. Iterated local search for the quadratic assignment problem. Technical report, aida-99-03, FG Intellektik, TU Darmstadt, 1999. (Citado en la página 1).

[4] F. Glover. Tabu search, 1989. (Citado en las páginas 2 y 26).

[5] F. Glover. Tabu search part ii. *Operations Research Society of America (ORSA) Journal on Computing*, 2(1):4–32, 1990. (Citado en las páginas 2 y 26).

[6] F. Glover and M. Laguna. Tabu search, 1997. (Citado en las páginas 2 y 26).

[7] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983. (Citado en las páginas 2, 14 y 56).

[8] E.H.L. Aarts, J.H.M. Korst, and P.J.M. Laarhoven. *Simulated Annealing*, pages 91–120. Local Search in Combinatorial Optimization. Wiley Interscience, Chichester, UK, 1997. (Citado en las páginas 2 y 56).

[9] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Reading, 1989. (Citado en las páginas 2 y 16).

[10] M. Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press,Cambridge, 1998. (Citado en las páginas 2 y 16).

[11] I. Rechenberg. *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, 1973. (Citado en las páginas 2 y 16).

[12] Swarm intelligence, August 2011. (Citado en las páginas 3 y 7).

[13] M. Dorigo and L. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. In *IEEE Transaction on Evolutionary Computation*, volume 1, pages 53–66, 1997. (Citado en la página 3).

[14] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE Service Center, Piscataway, NJ, 1995. (Citado en las páginas 3, 8 y 17).

[15] K. A. DeJong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975. (Citado en la página 3).

[16] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In J. J Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, pages 41–49, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc. (Citado en la página 3).

[17] R. Battiti. *Reactive search: Toward selftuning heuristics*, pages 61–83. Modern Heuristic Search Methods. Wiley, Chichester, UK, 1996. (Citado en la página 4).

[18] T. Stützle. Local search algorithms for combinatorial problems-analysis, algorithms and new applications. *DISKI-Dissertationen zür Kunstliken Intelligenz*, 1999. (Citado en la página 4).

[19] H. Sheng-Ta, S. Tsung-Ying, L. Chan-Cheng, and T. Shang-Jeng. Solving large scale global optimization using improved particle swarm optimizer. In *Proceedings of the 2008 IEEE Congress on Evolutionary Computation*, pages 1777–1784. IEEE Press, 2008. (Citado en las páginas 4 y 21).

[20] S. Z. Zhao, J. J. Liang, P. N. Suganthan, and M. F. Tasgetiren. Dynamic multi-swarm particle swarm optimizer with local search for large scale global optimization. In *Proceedings of the 2008 IEEE Congress on Evolutionary Computation*, pages 3845–3852. IEEE Press, 2008. (Citado en las páginas 4 y 23).

[21] T. Hendtlass. Wosp: A multi-optima particle swarm algorithm. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 727–734, Edinburgh, UK, 2005. IEEE. (Citado en las páginas 4 y 24).

[22] S. Chen. Locust swarms - a new multi-optima search technique. In *Proccedings of the 2009 IEEE Congress on Evolutionary Computation*, pages 1745–1752, Trondheim, Norway, 2009. IEEE. (Citado en las páginas 4, 23, 26, 28, 29, 31 y 33).

[23] M. A. Muñoz, J. A. López, and E. F. Caicedo. Inteligencia de enjambre: Sociedades para la solución de problemas (una revisión), August 2008. (Citado en la página 7).

[24] J. Kennedy and R. C. Eberhart. A discrete binary version of the particle swarm algorithm. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 5, pages 4104–4108, 1997. (Citado en la página 7).

[25] M. Clerc. Discrete particle swarm optimization illustrated by the traveling salesman problem, February 2000. (Citado en la página 7).

[26] W. Zhong, J. Zhang, and W. Chen. A novel discrete particle swarm optimization to solve traveling salesman problem. In *IEEE Congress on Evolutionary Computation*, pages 3283–3287. IEEE, September 2007. (Citado en la página 7).

[27] F. Heppner and U. Grenander. A stochastic nonlinear model for coordinated bird flocks. In S. Krasner, editor, *The ubiquity of chaos*, pages 233–238. AAAS Publications, Washington, DC, 1990. (Citado en la página 8).

[28] E. O. Wilson. *Sociobiology: The new synthesis*. Belknap Press of Harvard University Press, Cambridge, MA, 1975. (Citado en la página 8).

[29] M. M. Millonas. Swarms, phase transitions and collective intelligence. In C. G. Langton, editor, *Artificial Life III*. Addison-Wesley, 1994. (Citado en la página 10).

[30] D. Bratton and J. Kennedy. Defining a standard for particle swarm optimization. In *Proceedings of the 2007 IEEE Swarm Intelligence Symposium (SIS 2007)*, pages 120–127, 2007. (Citado en las páginas 11, 14, 15, 16, 19, 20, 44 y 55).

[31] E.G. Talbi. *Metaheuristics: from design to implementation*. Wiley Series on Parallel and Distributed Computing. John Wiley & Sons, 2009. (Citado en las páginas 12, 14 y 16).

[32] Y. Shi and R. C. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE World Congress on Computational Intelligence, the IEEE International Conference on Evolutionary Computation*, pages 69–73, Anchorage, AK or Piscataway, NJ, USA, 1998. IEEE Press. (Citado en la página 14).

[33] V. Cerny. Thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985. (Citado en la página 14).

[34] Y. Shi and R. C. Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation, CEC'1999*, volume 3, pages 1945–1950, Piscataway, NJ, USA, 1999. IEEE Press. (Citado en las páginas 14 y 15).

[35] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002. (Citado en la página 15).

[36] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, 2007. (Citado en las páginas 17, 18 y 19).

[37] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science MHS '95*, pages 39–43, Nagoya, Japan, 1995. IEEE Service Center, Piscataway, NJ. (Citado en las páginas 17, 19 y 55).

[38] J. Rada-Vilela, M. Zhang, and W. Seah. A performance study on synchronous and asynchronous updates in particle swarm optimization. In

Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11, pages 21–28, New York, NY, USA, 2011. ACM. (Citado en las páginas 18, 19 y 75).

[39] A. Engelbrecht. *Computational Intelligence: An introduction*. John Wiley & Sons Ltd, 2 edition, 2007. (Citado en la página 18).

[40] Mutation operators, August 2011. (Citado en la página 22).

[41] L. Cartwright and T. Hendtlass. A heterogeneous particle swarm. In K. Korb, M. Randall, and T. Hendtlass, editors, *Fourth Australian Conference on Artificial Life (ACAL)*, LNAI 5865, pages 201–210. Springer-Verlag, Berlin Heidelberg, 2009. (Citado en la página 29).

[42] M. A. Montes de Oca, J. Peña, T. Stützle, C. Pinciroli, and M. Dorigo. Heterogeneous particle swarm optimizers. Technical Report TR/IRIDIA/2009-001, INRIA, Université Libre de Bruxelles, Bruxelles, Belgium, January 2009. (Citado en la página 29).

[43] S. Chen. An analysis of locust swarms on large scale global optimization problems. In K. Korb, M. Randall, and T. Hendtlass, editors, *Proceedings of Fourth Australian Conference on Artificial Life (ACAL)*, LNAI 5865, pages 211–220. Springer-Verlag, Berlin Heidelberg, 2009. (Citado en las páginas 29, 30 y 33).

[44] S. Chen and Y. N. Vargas. Improving the performance of particle swarms through dimension reductions - a case study with locust swarms. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, pages 2950–2957, Barcelona, Spain, 2010. IEEE. (Citado en las páginas 29, 30, 33 y 50).

[45] Random search, August 2011. (Citado en la página 29).

[46] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical report, Caltech Concurrent Computation Program, 1989. (Citado en las páginas 29 y 56).

[47] Memetic algorithm, August 2011. (Citado en la página 29).

[48] S. Finck, N. Hansen, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Technical Report 2009/20, Research Center PPE, Austria, 2009. (Citado en las páginas 29, 30, 41 y 62).

[49] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, and Z. Yang. Benchmark functions for the CEC 2008 special session and competition on large scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2007. (Citado en las páginas 30 y 31).

[50] D. Kalyanmoy. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley & Sons, LTD, Department of Mechanical Engineering. Institute of Technology, Kanpur, India, 2001. (Citado en la página 56).

[51] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. Technical Report 3, IRIDIA, Université Libre de Bruxelles, Bruxelles, Belgium, September 2003. (Citado en la página 56).

[52] Computación paralela, September 2011. (Citado en la página 74).

[53] Gpu - unidad de procesamiento gráfico, 2011. (Citado en la página 74).

[54] J. F. Schütte, J. A. Reinbolt, B. J. Fregly, R. T. Haftka, and A. D. George. Parallel global optimization with the particle swarm algorithm. *Journal of Numerical Methods in Engineering*, 61:2296–2315, 2003. (Citado en la página 75).

[55] G. Venter and J. S. Sobieski. A parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. *Journal of Aerospace Computing, Information and Communication*, 2005. (Citado en la página 75).

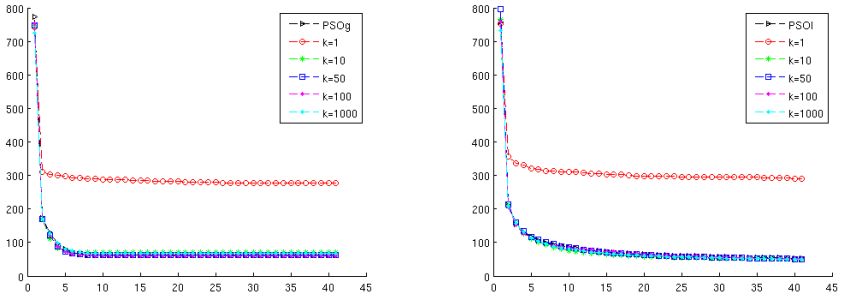
[56] B. Koh, A. D. George, R. T. Haftka, and B. J. Fregly. Parallel asynchronous particle swarm optimization. In *International Journal for Numerical Methods in Engineering*, volume 67, pages 578–595, 2006. (Citado en la página 75).

[57] L. Mussi, Y. S. G. Nashed, and S. Cagnoni. Gpu-based asynchronous particle swarm optimization. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 1555–1562, New York, NY, USA, 2011. ACM. (Citado en las páginas 75 y 76).

[58] Y. N. Vargas and S. Chen. Particle swarm optimization with resets – improving the balance between exploration and exploitation. In G. et al. Sidorov, editor, *Proceedings 9th Mexican International Conference on*

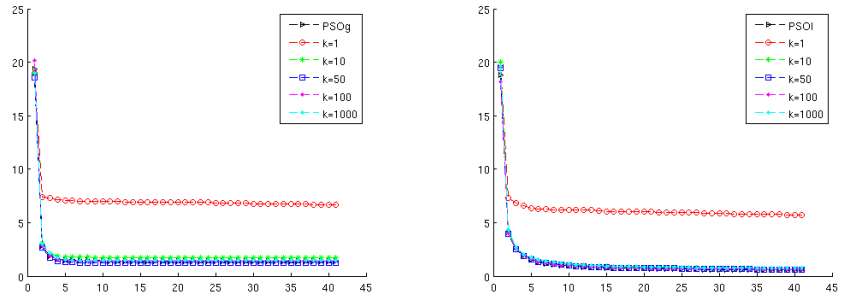
Artificial Intelligence, number 2 in MICAI 2010, pages 371–381, Pachuca, Mexico, 2010. Springer-Verlag Berlin Heidelberg 2010. (Citado en la página 78).

Anexos



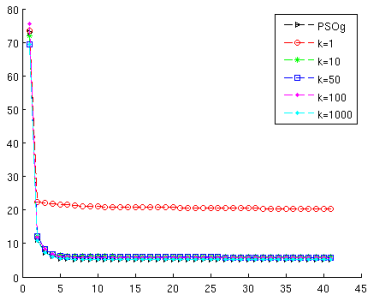
(a) Versión global en F15 (b) Versión local en F15

Figura 3.4: Convergencia de PSO estándar y R-PSO para diferentes valores de k en F15.

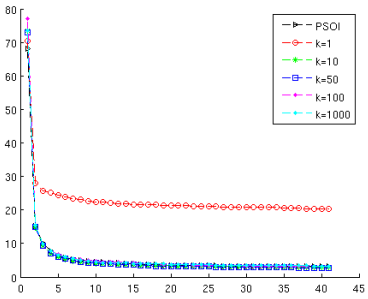


(a) Versión global en F17 (b) Versión local en F17

Figura 3.5: Convergencia de PSO estándar y R-PSO para diferentes valores de k en F17.

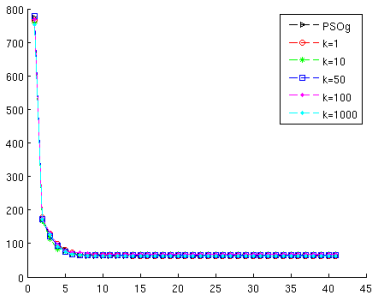


(a) Versión global en F18

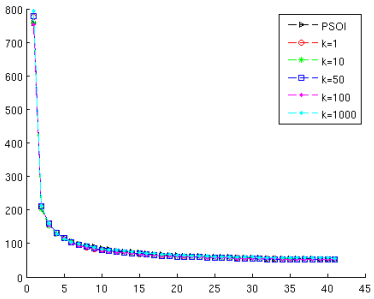


(b) Versión local en F18

Figura 3.6: Convergencia de PSO estándar y R-PSO para diferentes valores de k en F18.

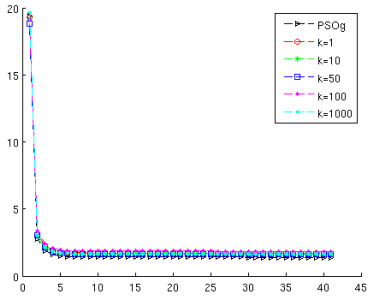


(a) Versión global en F15

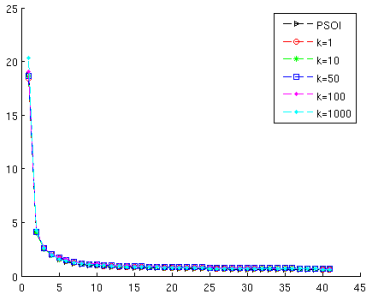


(b) Versión local en F15

Figura 3.7: Convergencia de PSO estándar y RE-PSO para diferentes valores de k en F15.

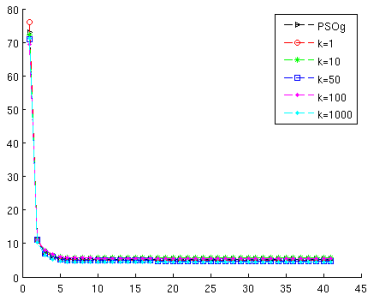


(a) Versión global en F17

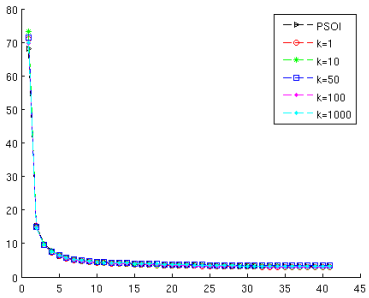


(b) Versión local en F17

Figura 3.8: Convergencia de PSO estándar y RE-PSO para diferentes valores de k en F17.

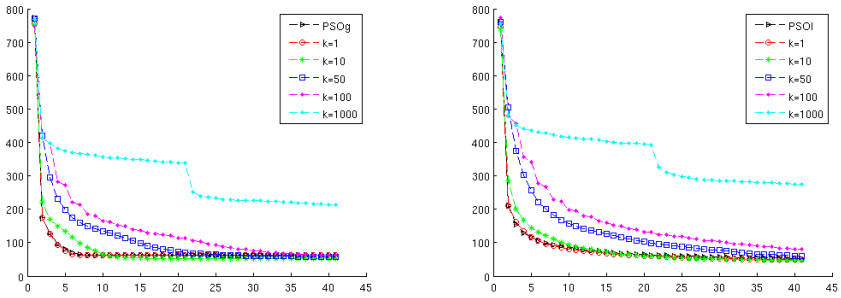


(a) Versión global en F18



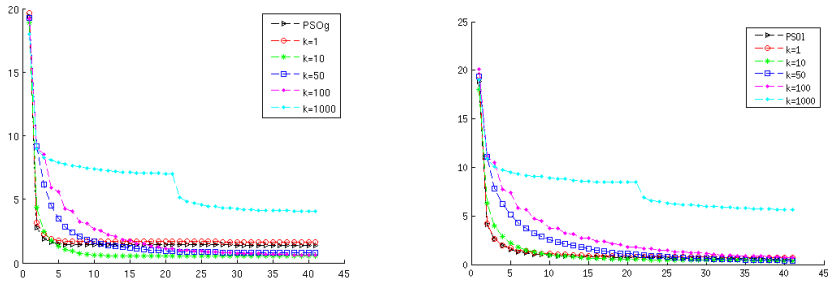
(b) Versión local en F18

Figura 3.9: Convergencia de PSO estándar y RE-PSO para diferentes valores de k en F18.



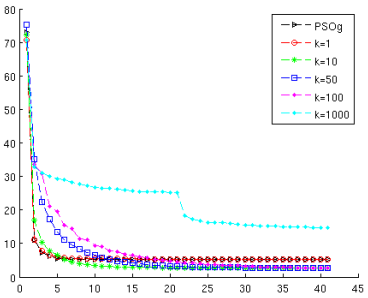
(a) Versión global en F15 (b) Versión local en F15

Figura 3.10: Convergencia de PSO estándar y AR-PSO para diferentes valores de k en F15.

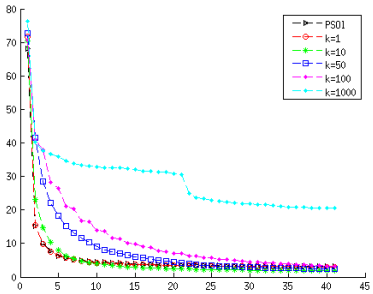


(a) Versión global en F17 (b) Versión local en F17

Figura 3.11: Convergencia de PSO estándar y AR-PSO para diferentes valores de k en F17.



(a) Versión global en F18



(b) Versión local en F18

Figura 3.12: Convergencia de PSO estándar y AR-PSO para diferentes valores de k en F18.