Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Un algoritmo micro-poblacional basado en cúmulos de partículas para problemas de optimización de alta dimensionalidad

Tesis que presenta

Jorge Salinas Lara

para obtener el Grado de

Maestro en Ciencias en Computación

Director de la Tesis:

Carlos Artemio Coello Coello

Ciudad de México **Noviembre, 2018**

Resumen

Existen muchos problemas de optimización global del mundo real que contienen un gran número de variables (100 o más) las cuales se definen mediante números reales. A éstos se les denomina "problemas de optimización global de gran escala" y se caracterizan por tener un espacio de búsqueda muy grande. Estos problemas son muy difíciles de resolver mediante técnicas de programación matemática debido a que suelen contar con un gran número de óptimos locales en los que los algoritmos pueden quedar fácilmente atrapados. Adicionalmente, la solución de estos problemas suele involucrar un costo computacional considerable, el cual crece con la dimensionalidad del problema. Estas limitantes han motivado el uso de técnicas alternativas de búsqueda y optimización, de entre las que destacan las metaheurísticas.

En esta tesis se propone el uso de una metaheurística bio-inspirada llamada "optimización mediante cúmulos de partículas" (PSO, por sus siglas en inglés) para optimización global de gran escala. PSO simula los patrones de vuelo de un grupo de aves que buscan comida y ha sido usada de manera exitosa en una amplia variedad de problemas de optimización global.

La principal contribución de esta tesis es el desarrollo de una versión del PSO que usa una población muy pequeña (no más de cinco partículas) para resolver problemas de optimización global de gran escala. El algoritmo propuesto (llamado micro-PSO o μ-PSO) es comparado con respecto a algoritmos del estado del arte en el área, utilizando problemas de prueba estándar de la literatura especializada.

Abstract

There are many real-world global optimization problems having a large number of decision variables (100 or more) which are defined through the use of real numbers. These are the so-called "large scale global optimization problems" and they are characterized for having a very large search space. These problemas are very difficult to solve using mathematical programming techniques, because they normally have a large number of local optima in which an algorithm can get easily trapped. Additionally, the solution to these problems normally involves a considerably high computational cost, which increases with the dimensionality of the problem. These limitations have motivated the use of alternative search and optimization techniques, from which metaheuristics are worth emphasizing.

In this thesis, we propose the use of a bio-inspired metaheuristic called "particle swarm optimization" (PSO) for large scale global optimization. PSO simulates the flight patterns of a flock of birds seeking for food, and it has been successfully used in a wide variety of global optimization problems.

The main contribution of this thesis is the development of a PSO version that adopts a very small population size (no more than five particles) to solve large scale global optimization problems. The proposed algorithm (called micro-PSO or μ-PSO) is compared with respect to state-of-the-art algorithms from the area, adopting standard test problems from the specialized literature.

Agradecimientos

Al CINVESTAV y en particular al departamento de computación, por brindarme la oportunidad de estudiar en esta gran institución.

Al CONACyT por sustentar económicamente mis estudios de maestría, permitiendome cerrar con éxito esta etapa de mi vida.

Este trabajo de tesis se derivó del proyecto Fronteras de la Ciencia 2016 titulado "Esquemas de Selección Alternativos para Algoritmos Evolutivos Multi-Objetivo" (Ref. 1920), cuyo responsable es el Dr. Carlos A. Coello Coello.

En primer lugar, quiero dar un agradecimiento muy especial a mis padres, Jorge Salinas y Esther Lara, por todo el apoyo moral, económico y emocional que me han dado no solo en la maestría, sino en todos mis proyectos.

A mi hermana Marisol, que me ha apoyado en la realización de mis objetivos y sin darse cuenta ha forjado gran parte de mi persona. Espero que esta tesis sea una motivación para ti en el cumplimiento de tus metas.

A mis abuelitos Hilario Salinas y Luis Lara, quienes me enseñaron a no rendirme, y sacar adelante toda meta que me propusiera, sé que estarían orgullosos de ver esta meta cumplida.

A mis abuelitas Modesta Lopez y Lucana Martinez, que han estado conmigo en todo momento, y ante cualquier problema, sé que puedo contar con ellas.

A Verónica Rodríguez, quien me apoyó moralmente en los buenos y malos momentos, motivandome en los momentos más tristes y compartiendo conmigo todo momento feliz, gracias por todo tu amor.

De forma muy especial, quisiera agradecer al Dr. Carlos Coello, por brindarme la oportunidad de trabajar bajo su supervisión y por todos los comentarios y observaciones que fueron sumamente importantes, tanto en el desarrollo de la tesis, como en el ámbito personal.

A Daybelis Jaramillo, quien ha sido una gran amiga para mi, siete años y contando, no dejaré de decirlo, es un gran camino que hemos compartido y aunque nuestros caminos se lleguen a separar, nuestra amistad seguirá siendo fuerte.

A Adrián Ramírez, quien me apoyó incondicionalmente a lo largo de la maestría, no solo en lo académico, sino en lo personal. Agradezco completamente tu amistad y estoy seguro que después de la maestría, seguirá creciendo.

A mis amigos de la maestría Manuel Hernández y Manuel Rodríguez, por cada momento que compartimos juntos en clases y en la sala de alumnos, trabajando o simplemente pasando el rato, son momentos que no se olvidan.

A mis amigos "Legales Ecatepec", por cada momento que hemos pasado juntos, demostrando que lo que para muchos puede ser un simple juego, para nosotros es una buena forma de pasar el tiempo.

A Santiago Domínguez por todo el apoyo que me brindó a lo largo de la maestría, no solo durante la elaboración de la tesis, sino a lo largo de mi formación académica.

Quiero agradecer de todo corazón a Sofy, por todo su apoyo y cariño hacia todos nosotros, quien más allá de cumplir con su obligación laboral, es como una mamá dentro de la escuela, siempre viendo por nuestro bienestar.

Índice general

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Antecedentes y motivación

Los problemas de optimización son comunes en diversas disciplinas del conocimiento. Cuando se considera un solo objetivo para ser optimizado, se busca encontrar la mejor solución disponible (llamada "óptimo global"), o al menos una buena aproximación de la misma. La optimización utilizando metaheurísticas se ha convertido en un tema popular de investigación en los últimos años [1].
Las metaheurísticas han demostrado tener excelente desempeño en problemas de alta complejidad, pero suelen perder su eficacia cuando abordan instancias de problemas con alta dimensionalidad (es decir, con más de cien variables de decisión) [2].
Dada la pérdida de eficacia de la mayoría de las metaheurísticas, existe el interés en diseñar un algoritmo de optimización utilizando una metaheurística que tenga la capacidad de obtener las mejores soluciones posibles de un problema de optimización de gran escala, sin que esto genere un incremento significativo en su costo computacional.

1.2. Planteamiento del problema

Muchos problemas del mundo real no pueden ser resueltos eficiente y/o eficazmente usando técnicas de programación matemática. Adicionalmente, el problema general de optimización no lineal no está resuelto, pues no existe ningún algoritmo que garantice encontrar el óptimo global en todos los casos. Esto ha motivado el desarrollo de las metaheurísticas, que son procedimientos de búsqueda de alto nivel que aplican una o varias reglas basadas en alguna fuente de conocimiento para explorar más eficientemente el espacio de búsqueda. Dentro de las muchas metaheurísticas existentes en la actualidad, las inspiradas biológicamente se han vuelto enormemente populares. La optimización mediante cúmulos de partículas (Particle Swarm Optimization, PSO) es una metaheurística bio-inspirada que se basa en los movimientos que realizan las parvadas de aves al buscar comida. PSO se ha vuelto muy popular debido a su simplicidad algorítmica y a su alta efectividad en un elevado número de problemas.
La micro optimización mediante Cúmulo de Partículas (Micro-PSO o μ-PSO) es una

versión del PSO que utiliza un tamaño de población muy pequeño (cinco partículas), y adopta un proceso de reinicialización para mantener la diversidad de la población. Adicionalmente incluye un operador de mutación para aumentar la capacidad de búsqueda del algoritmo [3]. Considerando que utiliza un tamaño de población pequeño, generalmente su costo computacional es menor debido a que se reduce el número de evaluaciones requeridas para alcanzar una solución de calidad aceptable [4].
Sin embargo, al intentar resolver un problema de optimización de alta dimensionalidad, puede ocurrir la llamada "maldición de dimensionalidad", es decir, conforme aumenta el tamaño espacio de búsqueda, el desempeño del algoritmo se deteriora. Las razones por las que este fenómeno ocurre son las siguientes:

- El espacio de solución de un problema usualmente se incrementa exponencialmente con la dimensión del problema, lo que implica que se requieren estrategias de búsqueda más eficientes para explorar todas las regiones prometedoras dentro de un presupuesto de tiempo dado.

- Las características de un problema pueden cambiar con la escala [2].

A pesar de ello, se pueden aprovechar las ventajas de la micro-población y la efectividad de PSO, para diseñar un algoritmo que permita resolver los problemas de alta dimensionalidad, de manera eficiente y eficaz.

1.3. Objetivos

1.3.1. General

Diseñar e implementar un algoritmo basado en PSO, que resuelva problemas de optimización mono-objetivo de alta dimensionalidad, utilizando una cantidad muy pequeña de partículas.

1.3.2. Particulares

- Investigar sobre las distintas metaheurísticas existentes que resuelvan problemas de alta dimensionalidad.

- Evaluar el algoritmo propuesto, tanto en el número de evaluaciones de la función objetivo que realice, como en el resultado final que obtiene, en problemas de optimización con diferentes dimensionalidades.

- Comparar el desempeño del algoritmo propuesto con metaheurísticas del estado del arte en optimización a gran escala.

1.4. Organización de la tesis

Esta tesis está formada por seis capítulos, y está organizada de la siguiente manera:
El capítulo dos tiene el objetivo de presentar los conceptos básicos de optimización global, e introducir los algoritmos evolutivos. Dichos conceptos brindan una guía al lector que no esté familiarizado con el tema.
El capítulo tres describe el algoritmo de optimización mediante cúmulos de partículas (PSO), en el cual está basada la propuesta de esta tesis. Inicialmente se introduce a grandes rasgos el algoritmo básico. Posteriormente se describen aspectos avanzados del mismo y después se habla de las topologías que puede utilizar, es decir, el cómo se comunican las partículas en un cúmulo. Por último se habla acerca del uso de poblaciones pequeñas en un PSO.
El capítulo cuatro habla acerca de la optimización de problemas de alta dimensionalidad. Aquí se describen los aspectos generales de este tipo de problemas, y posteriomente se discute el estado del arte en torno a la resolución de problemas de alta dimensionalidad.
El quinto capítulo detalla la propuesta dada en esta tesis para resolver problemas de alta dimensionalidad y se realiza la comparación de los resultados obtenidos al evaluar la propuesta.
El capítulo seis presenta las conclusiones de este trabajo de tesis, así como algunas posibles líneas de trabajo a futuro.

Capítulo 2

Conceptos básicos

Optimización es el proceso de obtener el mejor resultado posible bajo ciertas condiciones dadas.

2.1. Optimización global

Cuando un problema de optimización involucra un solo objetivo, a la búsqueda de la solución óptima se le denomina **optimización mono-objetivo** u **optimización global**.
Un problema de optimización mono-objetivo se puede definir matemáticamente de la siguiente manera:
Minimizar $f(\vec{x})$
sujeta a:

$$g_i(\vec{x}) \leq 0, \quad i = 1, \cdots, p$$

$$h_i(\vec{x}) = 0, \quad j = 1, \cdots, n$$

donde:
$\vec{x}$ es un vector de dimensión n llamado *vector de decisión*, $f(\vec{x})$ es llamada *función objetivo* $g_i(\vec{x})$ son las *restricciones de desigualdad*, $h_j(\vec{x})$ son las *restricciones de igualdad*.

El vector de decisión contiene las **variables de decisión**. Éstas modifican sus valores en un intervalo específico a lo largo de la resolución del problema. Los límites de este intervalo, denotados como límite inferior y límite superior, constituyen el espacio de las variables o el espacio de decisión S.
La **función objetivo** es la función que se va a optimizar y está definida en términos de las variables de decisión.
Si una solución no satisface las restricciones de igualdad y desigualdad o los límites de las variables, se considera que la *solución no es factible*. En caso de que se cumplan las restricciones y los límites de las variables, se le conoce como *solución factible*. Al conjunto de todas las soluciones factibles se le conoce como **región factible**.
Los problemas de optimización donde no existen restricciones, son denominados como

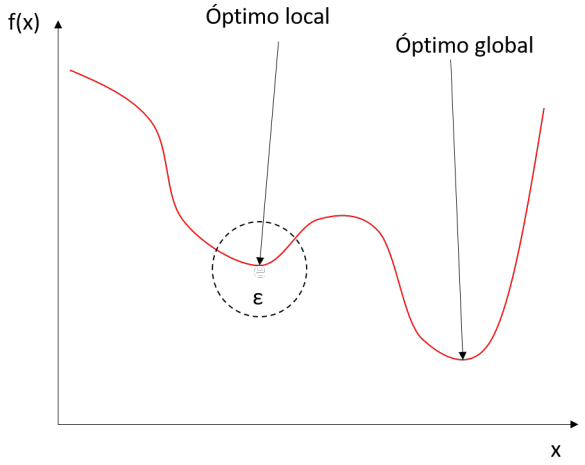

Figura 2.1: Ejemplo de una función de una variable con un mínimo local y un mínimo global

problemas de optimización sin restricciones, mientras que los problemas con un número de restricciones de igualdad o desigualdad mayor a cero, se les denomina problemas de optimización con restricciones.
Al optimizar una función, pueden existir una o varias soluciones factibles que sean mínimas o máximas, dependiendo de si se busca minimizar o maximizar, en la vecindad de algún vector de decisión, las cuales se denominan **óptimo local**. Si una solución factible es mínima y no existe una solución en todo el espacio de búsqueda que sea menor (en el caso de un problema de minimización), es conocida como **óptimo global**. Un ejemplo se puede observar en la figura 2.1.
La soluciones, denominadas *mínimo local* y *mínimo global*, se pueden definir como:

Definición 1 *Una función $f(\vec{x})$ posee un mínimo local en $\vec{x^l} \in \Omega$ si y sólo si $f(\vec{x^l} \leq f(\vec{x})$ para toda $\vec{x}$ a una distancia ϵ de $\vec{x^l}$.*

Es decir, existe una $\epsilon > 0$ tal que para toda $\vec{x}$ que satisface $\| \vec{x}.\vec{x^l} \| \in \epsilon$, $f(\vec{x^l}) \leq f(\vec{x})$.

Definición 2 *Dada una función $f : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$, $\Omega \neq \varnothing$ para $\vec{x} \in \Omega$, el valor $f^* \triangleq f(\vec{x^*}) > -\infty$ es llamado mínimo global si y sólo si $\forall \vec{x} \in \Omega : f(\vec{x^*}) \leq f(\vec{x})$.*

De esta manera, $f(\vec{x^*})$ es la solución (o soluciones) mínima(s) global(es), f es la función objetivo y el conjunto Ω es la región factible ($\Omega \subset S$).

2.1.1. Técnicas de optimización

Existen varias técnicas clásicas para resolver problemas con ciertas características específicas. Es importante saber al menos de la existencia de estas técnicas, pues cuando el problema por resolverse se adecúa a ellas, es innecesario el uso de heurísticas.
Para problemas de optimización lineal, el método Simplex es considerada la opción

más viable. Este método se basa en la observación, de que al encerrar el conjunto factible de un problema lineal en una especie de poliedro, entonces los valores máximos o mínimos de las funciones se encontrarán en un vértice [5].
Para optimización no lineal existen los métodos directos, tales como la búsqueda de patrones (Hooke y Jeeves), método simplex no lineal (Nelder Mead) o el método de direcciones conjugadas (Powell) [6]. Tambien existen métodos de búsqueda no directos, por ejemplo los llamados métodos de gradiente, tales como el descenso empinado (Cauchy) y el gradiente conjugado (Fletcher y Reeves) [7].
Uno de los problemas de las técnicas clásicas de optimización es que suelen requerir información que no siempre está disponible. Por ejemplo, métodos como el del gradiente conjugado requieren de la primera derivada de la función objetivo. Otros, como el de Newton, requieren además de la segunda derivada. Por lo tanto, si la función objetivo no es diferenciable, estos metodos no pueden aplicarse.
Cuando enfrentamos un cierto problema de optimización, si la función a optimizarse se encuentra definida en forma algebraica, es importante intentar resolverla primero con técnicas clásicas, antes de utilizar cualquier heurística.

2.2. Conceptos de computación evolutiva

El término **Computación Evolutiva** (CE) describe un área de investigación dentro de las ciencias de la computación, en el cual se estudian metaheurísticas inspiradas en la teoría de la evolución natural de Darwin. Dichas metaheurísticas reciben el nombre genérico de **Algoritmos Evolutivos** (AEs) [8].

2.3. Neo-Darwinismo

El paradigma Neo-Darwiniano está basado en la teoría evolutiva propuesta originalmente por Charles Darwin, en combinación con el seleccionismo de August Weismann y las leyes de la genética de Gregor Mendel [9].
La teoría de la evolución de Darwin ofrece una explicación de los orígenes de la diversidad biológica y está basado en dos principios básicos: El primero es la **selección natural**, la cual favorece a aquellos individuos que compiten por los recursos existentes en un ambiente dado. En otras palabras, favorece a los individuos que se adapten mejor a las condiciones ambientales. A este fenómeno se le conoce como la *ley del más apto*. El segundo principio se refiere a la existencia de pequeñas **variaciones fenotípicas** entre los miembros de la población. Los rasgos fenotípicos son aquellas características físicas y de comportamiento hereditarias de cada individuo, que afectan directamente su respuesta al ambiente y a los otros individuos, las cuales determinan su aptitud. Cada individuo representa una combinación única de rasgos fenotípicos que son evaluadas por un ambiente. Si estos rasgos resultan favorables, existe una mayor probabilidad de que el individuo se reproduzca, heredando sus características

(si éstas son hereditarias); de lo contrario queda descartado. Darwin observó también que existen pequeñas variaciones aleatorias a nivel fenotípico, llamadas mutaciones, las cuales ocurren durante la reproducción. Con estas mutaciones, surgen nuevas combinaciones de rasgos, las cuales son evaluadas en el ambiente, donde los individuos más aptos sobreviven y se reproducen [8].
La teoría del plasma germinal de Weismann (llamada también seleccionismo) describe dos tipos de tejidos: el plasma somático y el plasma germinal. El plasma somático forma la mayor parte del cuerpo de un individuo (fenotipo), mientras que el plasma germinal es una porción de un organismo que transmite la información hereditaria (genotipo).
La teoría de Mendel está basada en un conjunto de reglas o leyes respecto a la transmisión de información genética hereditaria durante la reproducción.

Los individuos y las especies pueden ser vistas como una dualidad de su código genético, el genotipo y el fenotipo. El genotipo provee un mecanismo de almacenamiento de evidencia de experiencia, de información históricamente adquirida [10]. Por otro lado, el fenotipo es representado a un bajo nivel por su genotipo.
El Neo-Darwinismo establece que la historia de toda la vida en nuestro planeta puede ser explicada a traves de un puñado de procesos estadísticos que actuan sobre y dentro de las poblaciones y especies [11]: la *reproducción*, la *mutación*, la *competencia* y la *selección*.

- La *reproducción* es una propiedad inherente de todas las formas de vida, dado que sin ésta, la vida propia no tendría forma de producirse. La reproducción permite la formación de nuevos individuos, llevando a cabo la transmisión de información genética hereditaria de los individuos a sus descendientes

- La *mutación* es una pequeña alteración o variación genética que ocurre durante la reproducción, produciendo nuevas combinaciones que pueden ser positivas. La mutación está garantizada en cualquier sistema que se reproduce y se encuentra en equilibrio.

- La *competencia* entre las especies surge como forma de adaptación y supervivencia entre las especies, dada la cantidad finita de recursos en un ambiente, y es la consecuencia del crecimiento poblacional en un ambiente finito.

- La *selección* es el resultado de la competencia entre los individuos y las especies, provocada por la limitación de recursos.

La *evolución* es el resultado de la interacción de estos cuatro procesos entre las poblaciones, a través de cada generación.

Dicho de este modo, la evolución se puede ver como un problema de optimización [12]. La selección aproxima los fenotipos a un valor tan cercano al óptimo como sea posible, con ciertas condiciones iniciales dadas y bajo ciertas restricciones ambientales.

2.4. Algoritmos Evolutivos

Existen varios tipos de AEs. La idea básica de todas estas técnicas es la misma: Dada una población de individuos en algún ambiente con recursos limitados, la competencia por esos recursos causa la selección natural (supervivencia del más apto). Esto provoca un incremento en la aptitud de la población. Dada una función de aptitud a ser maximizada, se puede crear un conjunto aleatorio de soluciones candidatas; esto es, elementos del dominio de la función. Después, se aplica la función de aptitud a estas soluciones como una medición abstracta de aptitud, donde entre más alta sea la aptitud, resulta mejor. Con base a estos valores de aptitud se seleccionan algunos de los mejores candidatos para formar la siguiente generación. Posteriormente, se aplica recombinación y/o mutación en ellos. La *recombinación* es un operador aplicado a dos o más candidatos seleccionados (padres), produciendo uno o más candidatos (hijos). La mutación es aplicada a un candidato, resultando en un nuevo candidato. Al ejecutar las operaciones de recombinación y mutación, se favorece la creación de un conjunto de nuevos candidatos (los hijos). Este proceso se repite hasta que un candidato con la suficiente calidad (una solución) sea encontrado o hasta alcanzar un cierto criterio de paro [8].

2.4.1. Elementos de un Algoritmo Evolutivo

Los componentes principales de un AE son [8] :

- Representación (definición de los individuos).
- Función de evaluación (función de aptitud).
- Población.
- Mecanismo de selección de padres.
- Operadores de recombinación y mutación.
- Mecanismo de supervivencia (reemplazo).

Para crear un algoritmo completo, es necesario especificar cada componente y definir su procedimiento de inicialización. De igual forma, si se desea detener el algoritmo en determinada etapa, se debe proveer una condición de paro.
El primer paso para definir un AE es relacionar el "mundo real" con el "modelo del AE", esto es, establecer un vínculo entre el contexto del problema original y el espacio de la solución del problema donde se realiza la evolución. Los objetos que forman las posibles soluciones en el contexto del problema original son denominados fenotipo, mientras que su codificación, los individuos en el AE, se denominan genotipo. La representación es la especificación de un mapeo entre el fenotipo y el genotipo. Por ejemplo, dado un problema de optimización donde las posibles soluciones sean enteros, el conjunto de enteros dados forman parte del fenotipo. En este caso, se pueden

representar los enteros usando números binarios. En tal caso, los números binarios serían el genotipo, y se requiere un mapeo adecuado entre ambos espacios.
El papel de la función de evaluación es representar los requisitos de una población, es decir, el objetivo a conseguir. En CE la función de evaluación es conocida como función de aptitud. El problema original a ser resuelto por un AE es un problema de optimización. En este caso, se utiliza el nombre de *función objetivo* es utilizado en el contexto del problema original, y la función de aptitud puede ser la función objetivo original o una transformación de la misma.
La población se encarga de mantener las posibles soluciones de un problema. La población forma la unidad de la evolución. En casi todos los AEs el tamaño de la poblacción es constante y no cambia en el tiempo. La selección (de padres) se realiza a nivel de la población. La *diversidad* de la población es una medida de la cantidad de soluciones distintas presentes.
La selección de padres en la población distingue entre los individuos con base en su calidad, y en particular, permite a los mejores individuos ser padres de la siguiente generación. La selección usualmente es probabilística. Los individuos con mejor calidad tienen más probabilidad de ser padres, aunque algunos individuos de baja calidad podrían ser seleccionados también; de lo contrario la población podría estancarse en un óptimo local.
Los operadores de recombinación y de mutación crean nuevos individuos a partir de los individuos existentes. En el caso de la recombinación, también conocida como cruza, se realiza una mezcla de los genotipos de dos padres en uno o dos genotipos que forman los descendientes. El principio básico es emparejar dos individuos con características diferentes pero favorables, para obtener un hijo que contenga ambas carcaterísticas.
Por otro lado, la mutación realiza una variación en el genotipo del individuo al que se le aplica el operador, esto con el fin de emular las pequeñas variaciones adaptativas de las especies a determinado ambiente.

2.5. Paradigmas de la computación evolutiva

Existen tres paradigmas principales en los algoritmos evolutivos, cuyas motivaciones y orígenes fueron independientes entre sí [13]:

- Estrategias evolutivas
- Programación evolutiva
- Algoritmos genéticos

Adicionalmente algunos autores consideran la programación genética como otro paradigma, aunque ésta suele verse como un tipo especial de algoritmo genético. A continuación describiremos brevemente cada uno de estos paradigmas.

2.5.1. Estrategias evolutivas

Las Estrategias Evolutivas (EE) fueron desarrolladas en 1964 por Ingo Rechenberg y Hans-Paul Schwefel quienes eran estudiantes de doctorado en la Universidad Técnica de Berlín.
La EE original fue llamada (1+1)-EE, debido a que consistía de un solo padre, el cual era mutado (es decir, sujeto a un cambio aleatorio) para producir un hijo. Después, el padre era comparado con su hijo y el mejor de ambos era seleccionado para ser el padre de la siguiente iteración (o generación) [13].
Rechenberg desarrolló una regla para ajustar la mutación tomando en cuenta la desviacion estándar, de modo que el algoritmo pudiera converger al óptimo global. Esta regla es conocida como"la regla de éxito 1/5" y dice que la razón de mutaciones exitosas con respecto al total de las mutaciones realizadas debe ser exactamente 1/5. Si es mayor, debe incrementarse la desviación estándar y si es menor, debe decrementarse, en ambos casos un 10 %. Schwefel desarrolló versiones poblacionales de la EE denominadas $(\mu+\lambda)-EE)$ y $(\mu,\lambda)-EE$. En la versión $(\mu+\lambda)-EE$, existen μ padres que producen λ hijos, los cuales se reducen a μ padres para la siguiente generación. Con esto, los padres pueden sobrevivir hasta que sean reemplazados por mejores hijos. La versión $(\mu,\lambda)-EE$ evita que todos los individuos puedan sobrevivir más allá de una generación, haciendo que solamente los hijos sean los que se conviertan en los nuevos padres [14].

El algoritmo básico de una $(1+1)-EE$ se presenta en el algoritmo 1.

Algoritmo 1 Algoritmo $1+1-EE$

Entrada: Población inicial, número de generaciones
Salida: Población evolucionada

1: Generar población inicial $G(0)$
2: **repeat**
3: Aplicar un operador de mutación a $G(0)$
4: Adaptar el tamaño de paso σ con la regla de éxito 1/5 de Rechenberg
5: Evaluar $G'(t)$
6: Reemplazar $G(t)$ con $G'(t)$, si mejora el resultado
7: **until** Se cumpla alguna condición de paro

Algunas características de las EE son las siguientes [8]:

- Las variables se manejan como vectores de números reales
- Utilizan recombinacón discreta o intermediaria
- Utilizan una mutación Gaussiana
- La selección de supervivientes suele ser determinista

- La selección de padres es determinística.

Adicionalmente una de las principales contribuciones de las EE es la **auto-adaptación**. Este mecanismo permite evolucionar tanto las variables del problema como los parámetros requeridos para la técnica.

2.5.2. Programación evolutiva

La programación evolutiva (PE) fue propuesta por Lawrence J. Fogel en los 1960s [15]. Su técnica consistía básicamente en hacer evolucionar autómatas de estados finitos, los cuales eran expuestos a una serie de símbolos de entrada (el ambiente), y se esperaba que, eventualmente, serían capaces de predecir las secuencias futuras de símbolos que recibirían. Fogel utilizo una función de "pago" que indicaba qué tan bueno era un cierto autómata para predecir un símbolo, y usó un operador basado en la mutación natural para efectuar cambios en las transiciones y en los estados de los automatas que tenderían a hacerlos más aptos para predecir secuencias de símbolos. El algoritmo básico de la programación evolutiva es muy parecido al de las estrategias evolutivas. La principal diferencia de la PE con las EE, está en su inspiración biológica: En PE cada individuo es visto como si fueran especies diferentes, por lo que no hay recombinación. Adicionalmente, los mecanismos de selección son diferentes. En PE cada padre genera exactamente un descendiente ($\lambda = \mu$), pero estos padres y las poblaciones de descendientes son combinadas y compiten en torneos estocásticos para sobrevivir. Normalmente en PE, se definen diversos operadores de mutación dependientes del problema.
El funcionamiento básico de la Programación Evolutiva, es mostrado el el algoritmo 2.

Algoritmo 2 Algoritmo básico de Programación Evolutiva

Entrada: Población inicial, número de generaciones, porcentaje de mutación
Salida: Población evolucionada

Generar población inicial $G(0)$
Evaluar $G(0)$
$t := 0$
repeat
    Aplicar un operador de mutación
    Evaluar $G(t)$
    Realizar selección mediante torneo estocástico
until Se cumpla alguna condición de paro

Las características principales de la PE, son las siguientes [8]:

- Suelen usar vectores de números reales para representar soluciones
- No utilizan recombinación
- Utilizan una mutación Gaussiana
- La selección de padres es determinística (cada padre crea un descendiente por medio de la mutación)
- La selección de supervivientes es probabilística, pues realizan torneos

Al igual que las EE, la PE utiliza autoadaptación sobre el operador de mutación.

2.5.3. Algoritmos genéticos

Los Algoritmos Genéticos (AGs) fueron desarrollados por John Holland a principios de los 1960s en la Universidad de Michigan [16]. La finalidad de su trabajo consistió en explicar el proceso de adaptación en los sistemas naturales y diseñar un sistema artificial el cual pudiese retener los mecanismos más importantes de los sistemas naturales. Aunque concebido originalmente en el contexto del aprendizaje de máquina, el algoritmo genético se ha utilizado mucho en optimización, siendo una técnica sumamente popular en la actualidad.
El AG también implementa la noción biológica de aptitud (la habilidad de supervivencia y reproducción) de un modo distinto al visto anteriormente. En los anteriores paradigmas se utiliza la aptitud para determinar qué descendiente sobrevivirá para estar en la siguiente generación; sin embargo, una vez que se encuentran en esa generación, no hay un criterio adicional respecto a qué individuos serán los que se van a reproducir; es decir, todos los padres tienen las mismas oportunidades [8].
Los AGs enfatizan la importancia de la recombinación sexual (que es su operador principal) sobre el operador de mutación (utilizado como operador secundario). Al igual que la programación evolutiva, utilizan una selección probabilística basada en aptitud.
Primero se genera una población inicial de forma aleatoria. Los individuos de esta población serán un conjunto de *cromosomas* o cadenas de caracteres (letras y/o números) que representen una posible solución a un problema. En su versión original, el AG codifica todo tipo de variables en binario [17]. Usando esta representación, al conjunto de bits de una cadena se les denomina *gene*, y el valor dentro de cada posición de un gene se le llama *alelo*, como se muestra en la figura 2.2.
Una vez que se haya elegido el tipo de codificación apropiado, se aplica una función de aptitud a cada uno de los cromosomas, para medir la calidad de la solución codificada por éste. Conociendo la aptitud de cada cromosoma, se realiza un proceso de selección para elegir los individuos que serán los padres de la siguiente generación (regulamente el que tenga mejor aptitud). Entre los mecanismos de selección más

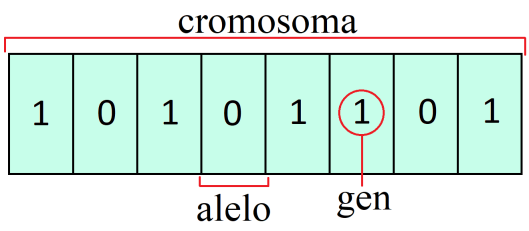

Figura 2.2: Ejemplo de uso de representación binaria de un cromosoma en un AG

empleados, están los siguientes [17]:

- **Selección proporcional:** Este nombre describe un grupo de esquemas de selección, donde se eligen los individuos con base en su contribución de aptitud respecto a toda la población.
- **Selección por jerarquías:** Los individuos se clasifican con base en su aptitud, y se les selecciona con base en su rango (o jerarquía) y no con base en su aptitud.
- **Selección mediante torneo:** La idea básica de este método es seleccionar con base en comparaciones directas de los individuos con base en sus valores de aptitud.
- **Selección de estado uniforme:** En esta técnica sólo uno o dos individuos son reemplazados en cada generación; generalmente se reemplazan los menos aptos.

Después de realizar el proceso de selección, se realiza la denominada *cruza* sexual. En esta etapa, se intercambia el material genético entre dos individuos, los cuales generan dos hijos. Existen tres operadores principales de cruza para codificación binaria:

- **Cruza de un punto:** En un punto n se fracciona la cadena binaria de dos cromosomas, y posteriormente intercambian fragmentos entre sí, como se muestra en la figura 2.3
- **Cruza de dos puntos:** En dos puntos m y n se fracciona la cadena binaria de dos cromosomas, y se intercambian fragmentos entre sí.
- **Cruza uniforme:** En este caso, se realiza una cruza de n puntos, pero no se tiene un número de puntos de cruza fijo.

Una vez realizada la cruza, se puede aplicar la *mutación*, como operador secundario, el cual cambia aleatoriamente uno o más alelos del cromosoma, como se muestra en la figura 2.4. Si se utiliza una representación binaria, una mutación cambia un 0 por 1 y viceversa. Este operador permite introducir nuevo material genético a la población y, desde una perspectiva teórica, asegura que, dada cualquier población, el espacio de búsqueda esté completamente conectado dado que la cruza no puede generar todas

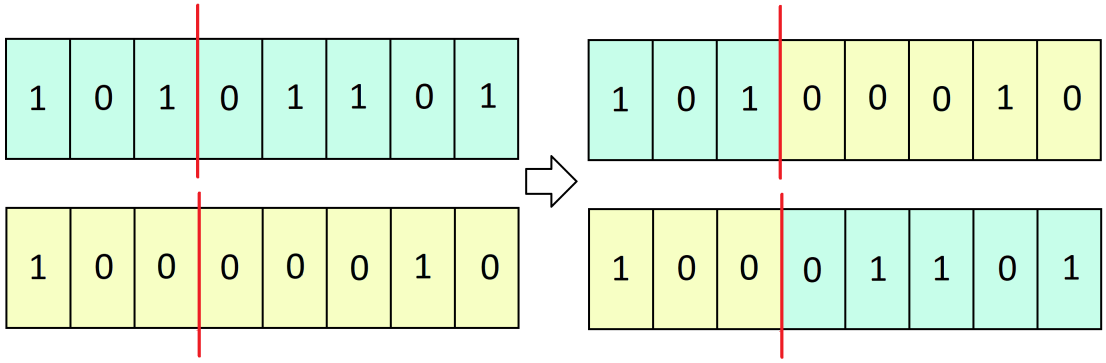

Figura 2.3: Ejemplo de cruza en un punto en un AG

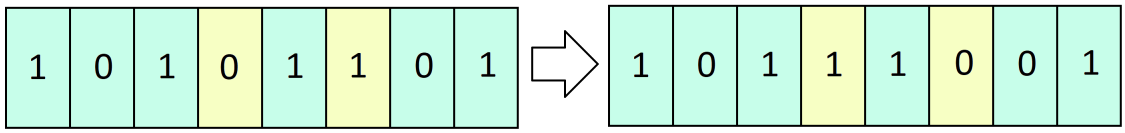

Figura 2.4: Ejemplo de mutación en un AG

las cadenas posibles [18].
El funcionamiento de un algoritmo genético simple se presenta en el algoritmo 3.

Algoritmo 3 Algoritmo Genético simple

Entrada: Población inicial, número de generaciones, porcentaje de cruza, porcentaje de mutación
Salida: Población evolucionada
Generar población inicial
Evaluar las aptitudes de la población inicial
repeat
  Seleccionar padres para ser recombinados
  Aplicar cruza a cada pareja de padres para producir la población de hijos
  Aplicar un operador de mutación a la población
  Evaluar la población que se vuelve la nueva población de hijos
until Se cumpla condición de paro

Se ha demostrado matemáticamente [19] que para converger al óptimo global se requiere mantener intacto al mejor individuo de cada generación. A este proceso se le conoce como **elitismo**.

2.5.4. Programación genética

Buscando evolucionar programas de computadora, Koza [20] sugirió el uso de un algoritmo genético con una codificación basada en árboles. Para simplificar la implementación de ese enfoque (llamado *programación genética*) fue hecho en LISP, aprovechando el hecho de que este lenguaje de programación tiene un analizador léxico integrado.
Los árboles de codificación adoptado por Koza permiten el uso de diversos alfabetos y operadores especializados para evolucionar programas generados aleatoriamente hasta lograr que los programas que se codifican sean 100 % válidos. Sim embargo, los principios básicos de esta técnica pueden ser generalizados hacia cualquier otro dominio, por lo cual la programación genética ha sido utilizada en una gran variedad de aplicaciones.
Los árboles utilizados en programación genética están formados por funciones y terminales. Las funciones normalmente adoptadas son las siguientes:

1. Operaciones aritméticas (ej.: +, -, ×, ÷).
2. Funciones matemáticas (ej.: seno, coseno, logaritmos, etc.).
3. Operaciones Booleanas (ej.: AND, OR, NOT).
4. Condicionales (IF-THEN-ELSE).

5. Ciclos (DO-LOOP).

6. Funciones recursivas.

7. Cualquier otra función de dominio específico.

2.5.5. Otras técnicas evolutivas

Recientemente se han diseñado nuevas técnicas evolutivas. Éstas no sólo se basan en la evolución natural, sino que algunas simulan procesos naturales. Como ejemplo, se pueden mencionar la Evolución Diferencial [21], los Sistemas Inmunes Artificiales [22], la Colonia de Hormigas [23] y la Optimización mediante Cúmulos de Partículas [24]. En el capítulo 3 se hablará con más detalle acerca de esta última, que es la metaheurística adoptada en esta tesis.

Capítulo 3

Optimización mediante cúmulo de partículas

El algoritmo de Optimización mediante cúmulo de partículas (*Particle Swarm Optimization* o PSO) fue propuesto en 1995 por James Kennedy y Russell Eberhart [24]. Es una metaheurística bioinspirada que se basa en el movimiento que realizan las aves para buscar alimento.

3.1. PSO visto de modo social y cognitivo

Un PSO puede ser visto desde un punto de vista social y cognitivo. En [25] se plantea una analogía con respecto a la transmisión de cultura entre los individuos de una población y la manera en que dicho mecanismo se puede usar para resolver problemas. Para ello plantea la existencia de una población de individuos, denominados agentes, los cuales interactúan entre sí con el objetivo de resolver un problema. Estos agentes poseen tres comportamientos principales los cuales se describen a continuación:

- **Evaluación:** Es la tendencia a catalogar cualquier estímulo como positivo o negativo. El aprendizaje no puede suceder al menos que el organismo pueda evaluar y distinguir las características del entorno. El aprendizaje puede ser definido como un cambio que permite al organismo mejorar la evaluación de su entorno.
- **Comparación:** Los agentes tienden a comparar sus características con respecto a otro agentes, y solamente imitan a aquellos vecinos que consideren mejores que ellos mismos, es decir, los que mejoren sus características, buscando con ello, mejorar la evaluación de su entorno.
- **Imitación:** Pocos animales en la naturaleza son capaces de realizar una imitación auténtica, siendo la imitación una forma efectiva de aprendizaje. Tomando esto en cuenta, un agente puede imitar a otro agente como una forma de aprendizaje.

Tomando esto en cuenta, PSO se puede ver como un modelo de aprendizaje social y cognitivo, donde un conjunto de agentes en movimiento buscan qué dirección seguir para encontrar la mejor ubicación posible. Para ello utilizan la mejor posición del recorrido, de acuerdo a la experiencia que ha tenido cada agente (evaluación) y a la mejor posición del recorrido de los demás agentes (comparación). Por último, cada agente se desplazará con base a tres direcciones: la dirección actual que tiene cada agente, la dirección guiada con base a la mejor posición de su experiencia propia y la dirección que considere la mejor de los demás agentes (imitación).

3.2. Descripción de la metaheurística

Un algoritmo de PSO está formado por los elementos que se definen a continuación:

- **Cúmulo:** Es equivalente a la población de individuos en un algoritmo evolutivo.
- **Partícula:** Es el individuo de un cúmulo. Cada partícula representa una solución posible al problema a resolver.
- **Factores de aprendizaje:** Están representados por c_1 (parámetro cognitivo) y c_2 (parámetro social). Estos parámetros definen la capacidad de búsqueda del algoritmo, donde el tener un valor elevado de c_1, incrementa la capacidad de búsqueda local de la partícula, y un valor elevado de c_2, incrementa la capacidad explorativa del algoritmo, con respecto a sus partículas vecinas.

Adicionalmente, cada partícula está formada por los siguientes elementos:

- Un vector $\vec{x}$ que define la posición actual de la partícula en el espacio de búsqueda. El tamaño está definido por el número de dimensiones/variables que tenga el problema que se va a resolver.
- Valor objetivo, usualmente definido como $f(\vec{x})$, que representa el valor de la solución de la partícula una vez que se ha evaluado la función objetivo del problema.
- La velocidad de la partícula $\vec{v}$. Representa la dirección que sigue la partícula en cada iteración del algoritmo para guiar su búsqueda y desplazarse en el espacio de soluciones.
- La mejor posición local $pBest$, es decir, el mejor valor de la función objetivo encontrado por la partícula.
- La mejor posición global $gBest$, que es el mejor valor de la función objetivo encontrado por todas las partículas del cúmulo.

La trayectoria de una partícula está definida por la velocidad de la partícula $\vec{v}$, la mejor posición conocida por la partícula y la mejor posición conocida de las

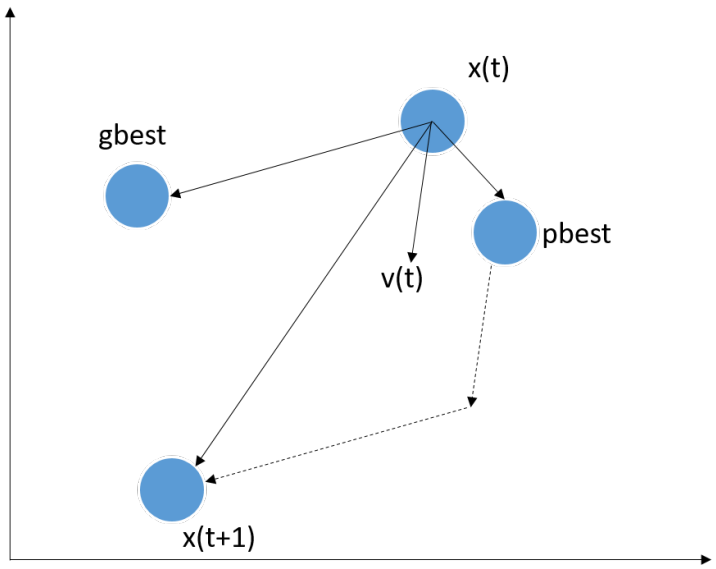

Figura 3.1: Representación gráfica de la trayectoria de una partícula

demás partículas del cúmulo. Estos tres elementos le permiten guiar su búsqueda a lo largo del espacio de soluciones, hacia las regiones en las que considere que existen las mejores posibilidades de encontrar una solución óptima, sin mantener una dirección de búsqueda fija.
Sea $\vec{x}_i(t)$ la posición actual de la partícula i en la generación t, y $\vec{v}_i(t)$ la velocidad asociada a la partícula i en la generación t. Para desplazar la partícula en la siguiente generación se utiliza:

$$\vec{x}_i(t+1) \leftarrow \vec{x}_i(t) + \vec{v}_i(t) \tag{3.1}$$

Sin embargo, como se mencionó anteriormente, la trayectoria se define utilizando un componente cognitivo y uno social. Para ello, la velocidad es modificada antes de actualizar la posición actual de la partícula, utilizando la siguiente ecuación:

$$\vec{v}_i(t+1) \leftarrow \vec{v}(t) + c_1 \cdot r_1 \cdot (pBest - \vec{x}_i) + c_2 \cdot r_2 \cdot (gBest - \vec{x}_i) \tag{3.2}$$

Donde r_1 y $r_2 \in [0, 1]$ y $\vec{v}(t)$ representa la dirección que tiene actualmente la partícula. El componente cognitivo está dado por $c_1 \cdot r_1 \cdot (pBest - \vec{x}_i)$, el cual representa la distancia entre la mejor posición conocida por la partícula y la posición actual; es decir, la experiencia de la partícula a lo largo de la trayectoria que ha recorrido. El componente social está dado por $c_2 \cdot r_2 \cdot (gBest - \vec{x}_i)$, y representa la distancia entre la mejor posición del cúmulo y la posición actual, o el aprendizaje de las partículas de su vecindario para guiar su búsqueda. Esto se puede ver como una suma de tres vectores para definir la nueva posición de la partícula, como se observa en la figura 3.1.
El algoritmo 4 muestra una versión básica de un PSO.

Algoritmo 4 Pseudocódigo de un algoritmo básico de PSO

Entrada: Número de partículas
Salida: La mejor solución encontrada

Inicializar aleatoriamente las posiciones $\vec{x}_i$ y $\vec{v}_i$ para las n partículas
Evaluar la función objetivo con las posiciones $\vec{x}$
Encontrar $pBest$
Encontrar $gBest$
while $t <$ número máximo de generaciones **do**
    for $i \leftarrow 1$ hasta n **do**
        Actualizar la velocidad utilizando
        $\vec{v}_i(t+1) \leftarrow \vec{v}(t) + c_1 \cdot r_1 \cdot (pBest - \vec{x}_i) + c_2 \cdot r_2 \cdot (gBest - \vec{x}_i)$
        Calcular las nuevas posiciones
        $\vec{x}_i(t+1) \leftarrow \vec{x}_i(t) + \vec{v}_i(t)$
        Evaluar las nuevas soluciones utilizando la función objetivo
    end for
    Encontrar $pBest$
    Encontrar $gBest$
    $t := t + 1$
end while
Reportar la mejor solución encontrada

3.3. Aspectos avanzados de PSO

Se han propuesto varias modificaciones al algoritmo básico de PSO, buscando que mejoren la calidad de las soluciones y la velocidad de convergencia. Dentro de estas modificaciones destacan los métodos para limitar la velocidad de las partículas. Esto surge debido a que las párticulas tienden a dirigirse hacia las regiones fuera del espacio de búsqueda. Entre estos métodos, se encuentran los siguientes: control de velocidad, inercia y factor de constricción. A continuación, se describe brevemente cada uno de ellos.

3.3.1. Control de velocidad

Es el método más simple, que consiste en emplear una constante llamada velocidad máxima v_{max}. De esta manera, en lugar de actualizar la velocidad con la ecuación 3.2, se emplea la siguiente:

$$\vec{v}_{i,j} = \begin{cases} \vec{v}_{i,j} & \text{si } \vec{v}_{i,j} \leq v_{max} \text{ y } \vec{v}_{i,j} \geq -v_{max} \\ v_{max} & \text{si } \vec{v}_{i,j} > v_{max} \\ -v_{max} & \text{si } \vec{v}_{i,j} < -v_{max} \end{cases} \tag{3.3}$$

Si es mayor el tamaño de la constante v_{max}, se incrementa la capacidad explorativa del algoritmo.

3.3.2. Inercia

Introducido por Yuhui Shi y Russell Eberhart [26], el factor de inercia es un parámetro que modifica la función de actualización de velocidad de las partículas. Tiene por objetivo que la velocidad no sobrepase los límites del espacio de búsqueda. Para ello, el factor de inercia (ω) multiplica la velocidad $\vec{v}$ en la ecuación (3.2). Este factor puede ser fijo o decrementarse después de cierto número de iteraciones. Con esto, la ecuación de actualización de velocidad queda de la siguiente manera:

$$\vec{v}_i(t+1) \leftarrow \omega \cdot \vec{v}(t) + c_1 \cdot r_1 \cdot (pBest - \vec{x}_i) + c_2 \cdot r_2 \cdot (gBest - \vec{x}_i) \tag{3.4}$$

Cuando ω toma valores pequeños ($\omega < 0.8$), el PSO tiene un funcionamiento similar a un buscador local. Si ω toma valores grandes, ($\omega > 1.2$), el PSO tiene un funcionamiento de buscador global, aumentando su capacidad explorativa e incluso explotativa en las nuevas áreas descubiertas. Shi y Eberhart sugieren que el uso de una inercia decremental de 1.4 a 0 es mejor que utilizar un factor de inercia fijo [27].

3.3.3. Factor de constricción

Maurice Clerc y James Kennedy [28] realizaron un análisis del movimiento de un cúmulo de partículas en un tiempo discreto y uno continuo, a partir del cual propusieron el uso de un coeficiente de constricción. El factor de constricción está formado por un conjunto de ecuaciones lineales, de cuya solución se obtiene una constante de constricción χ. Ésta es utilizada para evitar que la velocidad se eleve repentinamente, evitando así que rebase los límites del espacio de búsqueda. La ecuación de velocidad (3.2) queda reemplazada por:

$$\vec{v}_i(t+1) \leftarrow \chi \cdot (\vec{v}(t) + c_1 \cdot r_1 \cdot (pBest - \vec{x}_i) + c_2 \cdot r_2 \cdot (gBest - \vec{x}_i)) \tag{3.5}$$

Donde el factor χ se define como:

$$\chi = \frac{2 \cdot K}{\parallel 2 - \varphi - \sqrt{\varphi(\varphi - 4)} \parallel} \tag{3.6}$$

Siendo

$$\varphi = \varphi_1 + \varphi_2$$

$$\varphi \geq 4$$

$$\varphi_1 = c_1 \cdot r_1$$

$$\varphi_2 = c_2 \cdot r_2$$

donde $K \in [0, 1]$. El parámetro K controla la amplitud de la constricción. Si los valores de K son cercanos a cero, se tiene una convergencía rápida. Con valores de K cercanos a uno, se tiene una convergencia más lenta, pero se cuenta con una mayor capacidad de exploración.

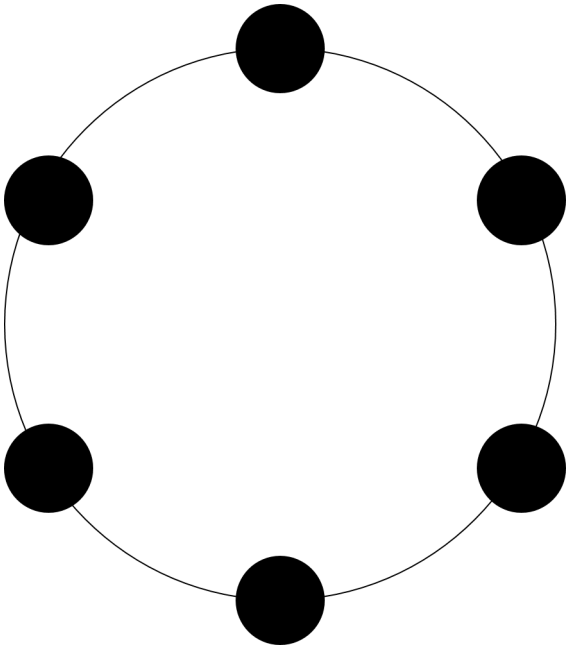

Figura 3.2: Representación gráfica de la topología lbest con seis partículas

3.4. Topologías

Las partículas tienden a beneficiarse de los valores de las partículas con las que están comunicadas. Una topología es la forma en la que se comunican las partículas en un cúmulo. Dentro de las topologías principales, se encuentran las siguientes: local, completa, estrella, árbol y von Neumann. A continuación se describe cada una de ellas.

3.4.1. Topología local

Cuando se usa la topología local, también conocida como lbest, cada partícula está conectada con sus vecinos inmediatos en el cúmulo. Su principal ventaja es que permite establecer subcúmulos que favorecen la exploración en las diversar regiones del espacio de búsqueda. Un ejemplo de esta topología se observa en la figura 3.2, donde se tienen seis partículas, donde todas están conectadas con dos vecinos inmediatos en el cúmulo, asemejando una topología de anillo.

3.4.2. Topología completa

También conocida como topología gbest, establece que toda partícula es vecina con todas las demás partículas del cúmulo, lo cual permite la explotación del espacio de búsqueda. Esta topología tiende a converger más rápido que la mayoría de las topologías, pero las partículas corren el riesgo de estancarse en un óptimo local. Un ejemplo se observa en la figura 3.3.

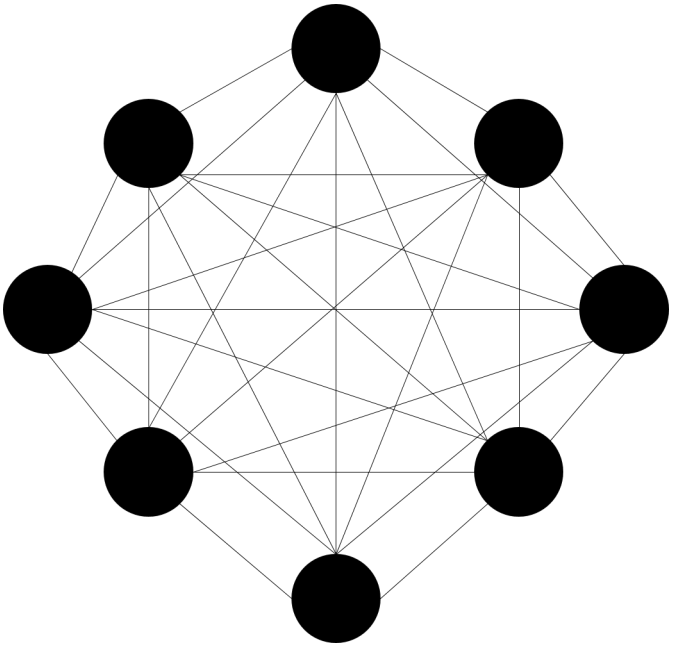

Figura 3.3: Representación gráfica de la topología gbest con siete partículas

3.4.3. Topología de estrella

Cuando se usa una topología estrella, una partícula se conecta a todas las demás, mientras el resto de las partículas se comunican solamente por medio de una partícula central. Ésta compara la información de las demás partículas del cúmulo, para actualizar la suya, y comunica el cambio al resto del cúmulo [29]. Un ejemplo de esta topología, se puede observar en la figura 3.4.

3.4.4. Topología de árbol

En esta topología, todas las partículas están ordenadas en una estructura de árbol, donde cada nodo del árbol contiene una partícula [30]. Cada partícula se ve influenciada por su mejor posición y la mejor posición de la partícula con la que esté conectada en un nivel superior del árbol (padre). Si la partícula ubicada en el nodo hijo, encuentra una mejor solución que la del nodo padre, ambas partículas son intercambiadas. Se puede observar un ejemplo de esta topología en la figura 3.5.

3.4.5. Topología von Neumann

La topología von Neumann, está formada por una matriz de tamaño $m \times n$, en la cual cada partícula está conectada con sus vecinos situados en el norte, sur, este y oeste.
Kennedy y Mendes [31] analizaron los efectos de distintas topologías y descubrieron que esta topología presenta mejores resultados que las topologías estándar en un con-

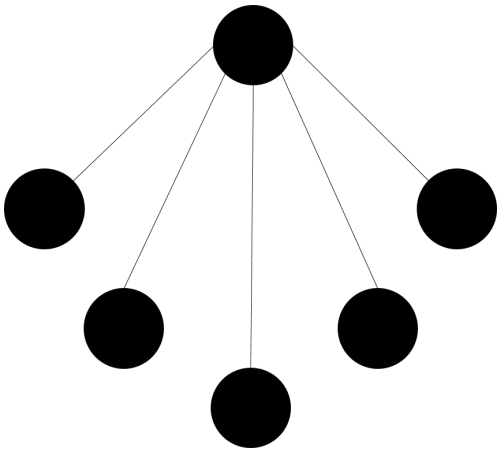

Figura 3.4: Representación gráfica de la topología de estrella con seis partículas

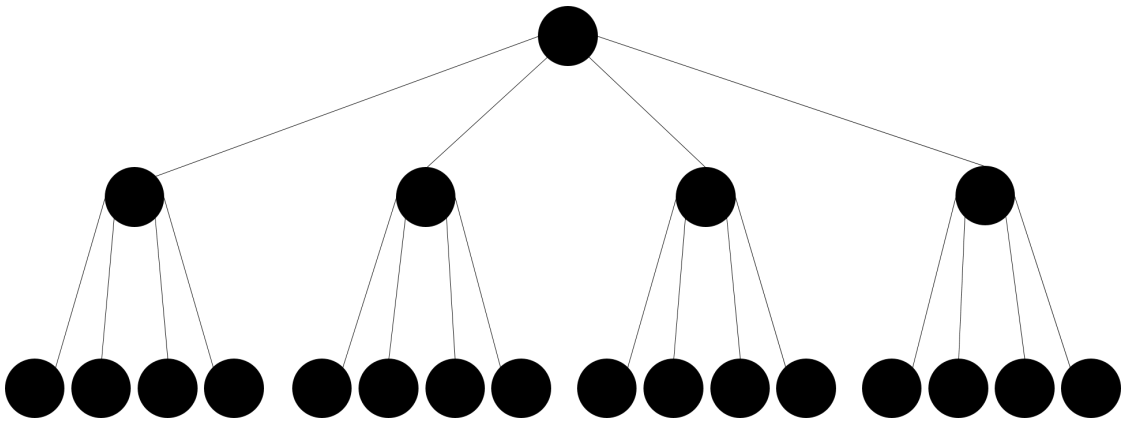

Figura 3.5: Representación gráfica de la topología de árbol

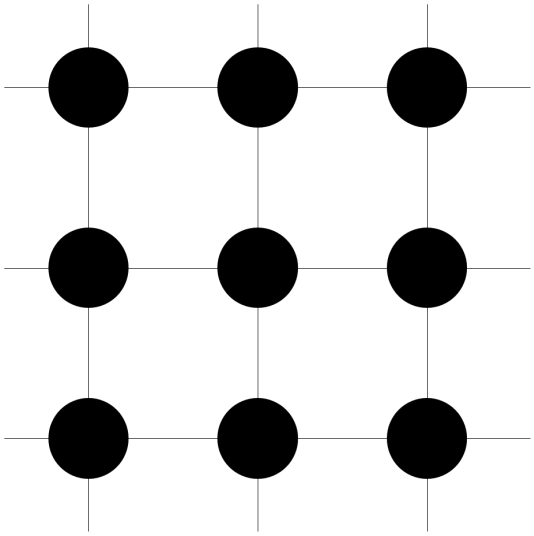

Figura 3.6: Representación gráfica de la topología von Neumann con nueve partículas

junto de problemas de prueba usados para optimización global.
Además, existe evidencia que sugiere que el uso de tamaños de población muy grandes es bueno en problemas de alta dimensionalidad y que las topologías altamente conectadas funcionan mejor en problemas unimodales, mientras que las topologías ligeramente conectadas son superiores en problemas muti-modales [32].

3.5. Micro Optimización mediante cúmulos de partículas

La micro Optimización mediante cúmulos de partículas (μ-PSO) se caracteriza principalmente por utilizar un tamaño de cúmulo muy pequeño, regularmente de cinco partículas. Sin embargo, este tipo de algoritmo se caracteriza por acelerar la pérdida de diversidad. Para evitarlo, Fuentes Cabrera [3] propone aplicar un proceso de reinicialización, lo cual mantiene la variedad de partículas del cúmulo. Para ello, se selecciona una cantidad de partículas de reemplazo, donde después de determinado número de iteraciones, se conservan las mejores soluciones encontradas, y se sustituyen las demás con soluciones aleatorias, permitiendo que se conserve la diversidad dentro del cúmulo. Adicionalmente, se emplea un operador de mutación, para mejorar la capacidad explorativa del algoritmo. El algoritmo 5 muestra una versión del μ-PSO

En la propuesta presentada en esta tesis, se utiliza la micro-población, en la búsqueda de reducir la cantidad de operaciones realizadas por el algoritmo. Adicionalmente se utiliza una búsqueda local en lugar de un proceso tradicional de reinicialización, mejorando así la capacidad explotativa del algoritmo propuesto.

Algoritmo 5 Pseudocódigo de un algoritmo básico de PSO

Entrada: Número de partículas, Número de soluciones para reemplazar NR, generación de reemplazo GR, c_1, c_2, probabilidad de mutación $PM \in [0, 1]$

Salida: La mejor solución encontrada

Inicializar aleatoriamente las posiciones $\vec{x}_i$ y $\vec{v}_i$ para las n partículas (máximo 6)
Evaluar la función objetivo con las posiciones $\vec{x}$
while $t <$ número máximo de generaciones **do**
    if $cont == GR$ **then**
        Reinicializar las NR peores soluciones
        $cont = 1$
    end if
    Encontrar $pBest$
    Encontrar $gBest$
    for $i \leftarrow 1$ hasta n **do**
        Actualizar la velocidad
        Calcular las nuevas posiciones
        Evaluar las nuevas soluciones con la función objetivo
    end for
    Aplicar operador de mutación con probabilidad PM
    $cont := cont + 1$
    $t := t + 1$
end while
Reportar la mejor solución encontrada

Capítulo 4

Optimización de problemas de alta dimensionalidad

Muchos problemas del mundo real se pueden plantear como problemas de optimización con variables en dominios continuos. Algunos ejemplos de esto, surgen en el área de diseño y control, tales como biotecnología, ingeniería industrial, procesamiento de señales, etc. Los algoritmos evolutivos han demostrado ser eficientes resolviendo este tipo de problemas.

En los últimos años, con mejores capacidades de procesamiento de cómputo, y con una gran cantidad de datos que se pueden analizar, han surgido nuevos retos. Uno de ellos es resolver problemas de optimización con un gran número de variables. A esta área se le conoce como **optimización global de gran escala** (*Large-scale global optimization*, LSGO).

Los problemas de LSGO se han convertido en un tema de interés en computación evolutiva, debido a que dichos problemas suelen presentarse en aplicaciones del mundo real. Sin embargo, el rendimiento de los algoritmos evolutivos en problemas de LSGO suele deteriorarse rápidamente; debido a que el espacio de búsqueda se incrementa considerablemente conforme se usan más variables. Esto conlleva a que se necesite realizar una búsqueda mucho más eficiente para explorar la mayor cantidad de soluciones posibles, sin que esto impacte significativamente en la cantidad de tiempo requerida [33].

En virtud de las dificultades que plantean los problemas de LSGO, los algoritmos diseñados para resolverlos tienen que ser más eficientes en la búsqueda de soluciones que aquellos diseñados originalmente para problemas de menor dimensionalidad. Esto ha motivado el diseño de mecanismos novedosos que permiten eficientar la búsqueda realizada por los algoritmos evolutivos en problemas de alta dimensionalidad.

4.1. Características de los problemas de optimización de alta dimensionalidad

Los problemas de optimización del mundo real suelen tener dependencias entre sus variables, principalmente cuando existe una gran cantidad de las mismas. Estos problemas contienen grupos de variables relacionadas entre ellas, entre los que se encuentran [34]:

Definición 3 *Separables: Una variable x_i es separable o no interactúa con otra variable si y sólo si:*

$$arg_x\ min\ f(x) = \left(arg_{x_i}\ min\ f(x), arg_{\forall x_j, j \neq i}\ min\ f(x)\right),$$

donde $x = (x_1, \cdots, x_D)^T$ es un vector de variables de decisión con D dimensiones. La notación $arg_x\ min\ f(x)$ significa que fue encontrado el valor óptimo de x_i, mientras que las demás variables se mantienen constantes.

Definición 4 *Parcialmente separables: Una función $f(x)$ es parcialmente separable con m subcomponentes independientes si y sólo si*

$$arg_x\ min\ f(x) = (arg_{x_1}\ min\ f(x), arg_{xm}\ min\ f(..., x_m)),$$

donde $x = (x_1, \cdots, x_D)^T$ es un vector de variables de decisión con D dimensiones, $x_1, \cdots, x_m$ son sub-vectores disjuntos de x, y $2 \leq m \leq D$.

Definición 5 *No separables: Son aquellas donde todas las variables de decisión del problema interactúan entre ellas.*

Definición 6 *Parcialmente aditivamente separables: Este es un caso especial de las funciones parcialmente separables, que representa de manera conveniente la mayoría de los problemas del mundo real y tiene la siguiente forma:*

$$f(x) = \sum_{i=1}^{m} f_i(x_i),$$

donde x_i son vectores de decisión mutuamente excluyentes de f_i, $x = (x_1, \cdots, x_D)^T$ es un vector de variables de decisión con D dimensiones, y m es el número de subcomponentes independientes.

Adicionalmente los problemas de LSGO, se pueden identificar de acuerdo a su modalidad. Una función es unimodal si sólo tiene un óptimo local, el cual a su vez es el óptimo global. Por otro lado, una función es multimodal, si tiene más de un óptimo local, pero sólo uno de ellos es el óptimo global.

4.2. Algoritmos para problemas de optimización de gran escala

Debido a la complejidad que tienen los problemas de optimización de gran escala, varios investigadores han trabajado en el desarrollo de algoritmos evolutivos que permitan resolver este tipo de problemas. Para ello, se han empleado técnicas co-evolutivas, que permiten la división de problemas grandes en problemas más pequeños, así como la escalarización de algoritmos diseñados para problemas de optimización global, los cuales fueron creados inicialmente para problemas de baja dimensionalidad. A continuación se presentan algunos algoritmos que se han desarrollado para problemas de optimización de gran escala.

4.2.1. Algoritmos co-evolutivos

Los algoritmos co-evolutivos se han utilizado como una estrategia para resolver problemas de optimización de gran escala. El enfoque principal utilizado por ellos, es la estrategia "divide y vencerás", mediante la cual se divide un problema de gran escala en varios problemas de baja dimensionalidad, con el objetivo de facilitar el proceso de optimización. Sin embargo, este tipo de algoritmos deben encontrar un proceso de división eficiente, pero dicha división depende del problema a resolverse [35].

DECC-G

En 2007, Yang et al. [36] propusieron un esquema de coevolución cooperativa (CC) con una estrategia de descomposición de problemas basada en grupos. La idea principal es separar un vector objetivo en m subcomponentes de s dimensiones de manera aleatoria, y evolucionar cada uno de ellos por medio de un algoritmo evolutivo. Con esto, plantean una estructura de agrupación dinámica, y utilizan pesos adaptativos para aplicar coadaptación entre los subcomponentes creados. Este esquema es combinado con un algoritmo de evolución diferencial: el algoritmo de evolución diferencial de búsqueda de vecindarios autoadaptativo (SaNSDE)[37] el cual es acoplado al esquema de CC propuesto.

MLCC

En 2008 Yang, et al. [38] propusieron un algoritmo llamado Coevolución cooperativa multinivel (MLCC). En este algoritmo, lo primero que se hace es diseñar diversos descomponedores de problemas basados en diferentes tamaños de grupos con la finalidad de formar un repositorio de descomponedores. Cada uno de los descomponedores denota distintos niveles de interacción entre las variables. Posteriormente, el proceso de evolución se divide en ciclos. Por cada ciclo, el algoritmo selecciona un descomponedor basado en sus registros de rendimiento, descompone el problema del vector

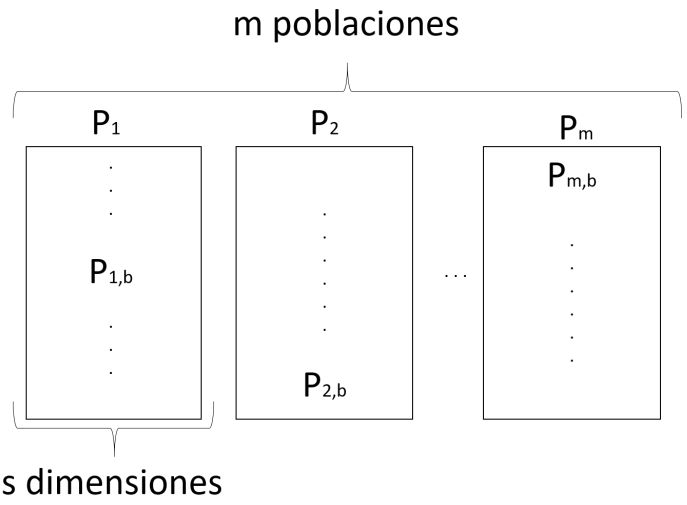

Figura 4.1: Representación gráfica de cómo se realiza la división en subproblemas. El vector objetivo n-dimensional se divide en m subcomponentes de s dimensiones. $P_{m,b}$ denota el mejor individuo del m-ésimo componente

objetivo en varios subcomponentes y los evoluciona con un algoritmo evolutivo determinado, proponiendo utilizar el algoritmo de evolución diferencial de búsqueda de vecindarios autoadaptativo (SaNSDE)[37]. Por último, se actualiza el registro de rendimiento del descomponedor utilizado.

DECC-ML

La coevolución cooperativa para optimización a gran escala a través de una agrupación aleatoria frecuente fue propuesta por Omidvar et al. [39] en 2010. La técnica que se propone en este caso es una optimización del desempeño de DECC-G (algoritmo de evolución diferencial basado en cooevolución cooperativa) y una alternativa al MLCC para la auto-adaptación de los tamaños de los subproblemas en los que divide un problema de alta dimensionalidad. Este esquema se puede observar con más detalle en la figura 4.1. Para evaluar el algoritmo, utilizan el conjunto de problemas de la competencia de optimización a gran escala realizada en el CEC 2008, utilizando 100, 500 y 1000 dimensiones.

CCPSO y CCPSO2

Li et al. [40] propusieron en 2009 un algoritmo llamado Optimización mediante cúmulo de partículas cooperativo y coevolutivo (CCPSO), donde se implementa una descomposición mediante agrupamiento aleatorio, en la que se pretende incrementar la probabilidad de que dos variables que interactúen entre sí estén en el mismo subcomponente. Se emplea una técnica llamada pesos adaptativos, donde se busca mejorar las soluciones propuestas. Con este esquema se logran resolver problemas de

optimización de hasta 1000 dimensiones.
En 2012, los mismos autores realizan una mejora de su algoritmo [41], en la cual proponen los siguientes cambios:

- Plantean un nuevo modelo de PSO, utilizando distribuciones de Cauchy y Gaussiana, para realizar muestreos alrededor de la mejor posición local.
- Utilizan la topología lbest en lugar de gbest, buscando resolver mejor las funciones multimodales.
- Eliminan los pesos adaptativos, y en su lugar proponen realizar más agrupamientos.
- Realizan pruebas con problemas de hasta 2000 dimensiones.

CCPSO-ISM

En 2014, se publicó un PSO competitivo y cooperativo con un mecanismo de intercambio de información para optimización global (CCPSO-ISM), propuesto por Li et al. [42]. La idea de este algoritmo es incorporar el mecanismo de intercambio de información (ISM), donde cada partícula comparte su mejor posición personal, en un "pizarrón" que solo conserva información actualizada de cada partícula, la cual utiliza para el operador competitivo y cooperativo. Adicionalmente, se integran dos parámetros nuevos, conocidos como "iteraciones estancadas" y "probabilidad de cooperación". Para evaluar el algoritmo, se emplearon los problemas de prueba de [43] Las pruebas se realizaron con 3, 5, 10, 15, 20, 30, 50 y 100 dimensiones.

SL-PSO

En 2017 fue publicado un algoritmo cooperativo basado en PSO para problemas con funciones objetivo costosas de alta dimensionalidad, propuesto por Sun et al. [44]. Este esquema utiliza métodos subrogados para ahorrar el consumo de tiempo en las evaluaciones de la función objetivo. El algoritmo propuesto está compuesto por un algoritmo basado en PSO con un factor de constricción [45], el cual realiza una estimación del valor de aptitud a nivel local y un algoritmo basado en PSO de aprendizaje social [46] que se encarga de la exploración. Adicionalmente, se utiliza un archivo para almacenar las partículas evaluadas utilizando la función de aptitud real, y una red de funciones de base radial [47]. Las pruebas se realizaron con problemas de prueba descritos en [48] [49], utilizando 50 y 100 dimensiones.

SWCC

En 2015, Can Liu y Bin Li [50] propusieron una búsqueda local mediante coevolución cooperativa basada en individuos para optimización a gran escala. Este esquema se basa en el algoritmo de Solis y Wets [51], que es una versión clásica de un Hill-climber, al cual le es incorporada una estrategia de coevolución cooperativa. Las

evaluaciones de prueba se realizaron con el conjunto de problemas del CEC 2013 [52], utilizando 1000 dimensiones, a excepción de las funciones F13 y F14 que utilizan sólo 905.

4.2.2. Variantes de PSO

Las estrategias coevolutivas han demostrado ser eficientes para resolver problemas de optimización de gran escala, pero no son la única estrategia empleada para conseguir este objetivo. Algunos autores han propuesto variantes de alguna metaheurística, particularmente de PSO, como se describe a continuación.

CSO

En 2013, Cheng et al. [53] propusieron un algoritmo basado en PSO, donde no se utilizan ni la mejor posición local ni la mejor posición global, llamado Optimizador mediante cúmulos competitivo (CSO). Las diferencias más notables entre este algoritmo y el PSO tradicional son las siguientes:

- En el PSO canónico, la búsqueda de soluciones está guiada por la mejor posición global (gbest) y la mejor posición individual (pbest), mientras que en el CSO no se utilizan estos conceptos. En cambio, se utiliza un mecanismo de competencia aleatorio, donde cualquier partícula puede ser un líder potencial, como se puede observar en la figura 4.2.
- En PSO se cuenta con un registro histórico de las mejores posiciones, mientras que en CSO no se aprende, sino que las partículas que pierden la competencia, aprenden de las partículas ganadoras en el cúmulo actual.

EPUS-PSO

Propuesto por Hsieh et al. [54] en 2009, consiste de una variante de PSO llamada Estrategia de utilización de población eficiente para PSO (EPUS-PSO), en la cual destaca el uso de un gestor de población para mejorar significativamente la eficiencia del PSO. Para ello utilizan partículas variables en los cúmulos para mejorar la habilidad de búsqueda y guiar las partículas de forma más eficiente. Adicionalente, se aplican los principios de compartir la solución entre las partículas, para evitar que éstas se estanquen en algún óptimo local, facilitando la búsqueda de un óptimo global.

DMS-PSO

En 2008, Zhao et al. [55] propusieron un algoritmo de optimización mediante cúmulos de partículas con multi-cúmulos dinámicos utilizando búsqueda local, adoptando una nueva topología de vecindarios. Para disminuir la velocidad de convergencia e incrementar la diversidad en el cúmulo para tener mejores resultados ante problemas multimodales, se utilizan vecindarios pequeños para resolver problemas.

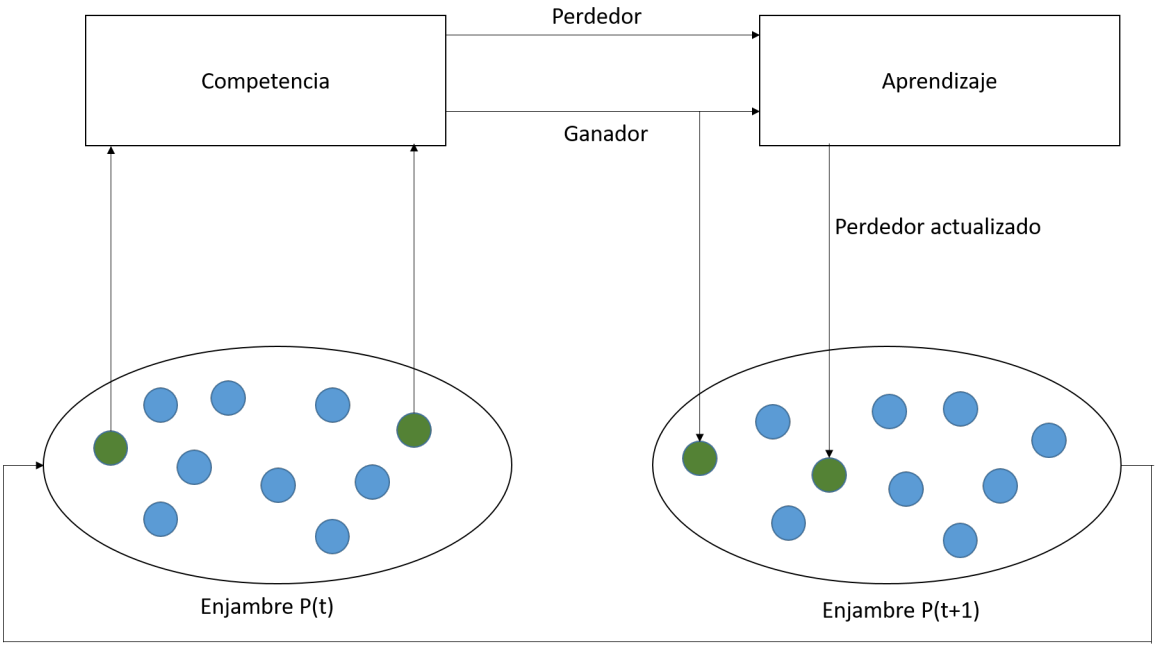

Figura 4.2: Representación gráfica del funcionamiento del torneo implementado en CSO

La población se divide en cúmulos de tamaño pequeño, y cada sub-cúmulo utiliza sus propias partículas para buscar la mejores regiones del espacio de búsqueda. Cada cúmulo utiliza su propia información histórica, facilitando la convergencia hacia un óptimo local. Sin embargo, se permite intercambiar la mayor cantidad de información entre las partículas para mejorar la diversidad. Adicionalmente, se incluye una reagrupación aleatoria para hacer que las partículas tengan estructuras de vecindarios que cambien dinámicamente. Por tanto, después de un determinado número de generaciones, la población se reagrupa aleatoriamente y comienza a buscar utilizando nuevos sub-cúmulos. Se puede observar el comportamiento de búsqueda de este algoritmo en la figura 4.3

CPSO-SL

En 2010, Liang et al. [56] propusieron un algoritmo de PSO cooperativo utilizando aprendizaje de interdependencia de variables estadísticas (CPSO-SL). Esta propusta se caracteriza principalmente por lo siguiente:

- Los subproblemas están traslapados y cada uno tiene un núcleo.
- Cada subproblema es optimizado por separado utilizando cada PSO individual.
- Se utiliza una solución global como memoria compartida, de tal forma que los PSOs pueden tomar variables y resultados optimizados.

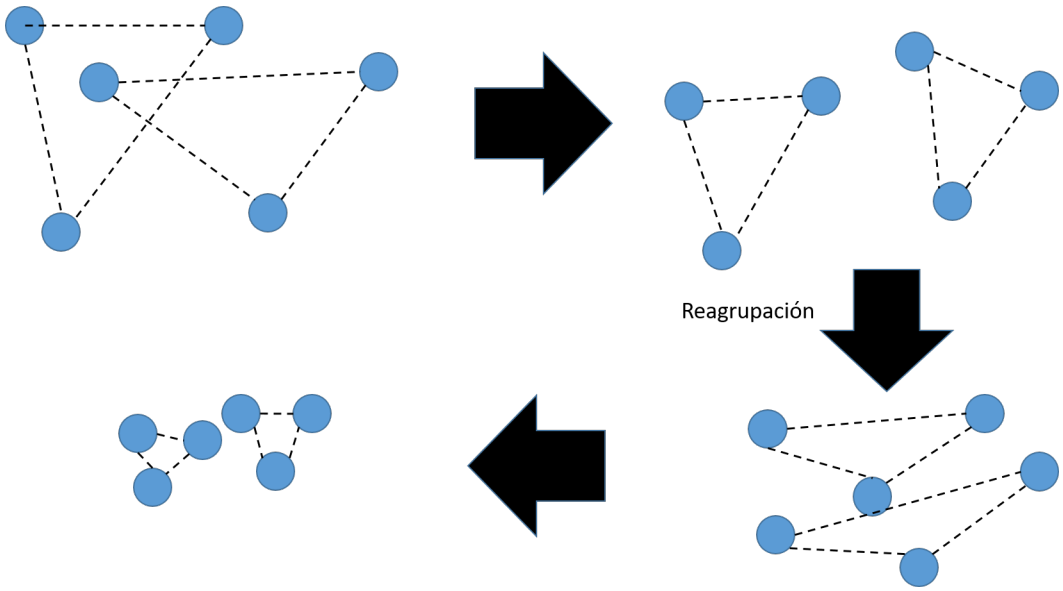

Figura 4.3: Representación gráfica de la búsqueda que realiza el DMS-PSO

hJPSO-VNS

Seren [57] propuso en 2011 un método basado en PSO para problemas discretos de alta dimensionalidad sin restricciones. En esta propuesta utilizan una hibridización entre PSO, los principios de la teoría de Variable Neighbourhood Search [58], en la cual no se sigue una trayectoria como normalmente ocurre con los métodos de búsqueda local, sino que se exploran incrementalmente los vecindarios distantes de la solución actual y el uso de *stretching* [59], que consiste en una transformación que modifica los valores de la función objetivo f en una región más o menos extensa alrededor de x^l, usualmente compuesta por puntos con la peor aptitud. En este artículo resuelven problemas de optimización sin restricciones de tipo combinatorio, usando instancias con 30, 500 y 1000 dimensiones. Este algoritmo tiene un buen funcionamiento para problemas combinatorios, pero requiere una discretización del espacio de búsqueda para resolver problemas continuos.

GPSO-PG

En 2016, Guo et al. [60] propusieron una variante de un algoritmo basado en PSO, utilizando agrupamiento con guía de P_{best} para optimización a gran escala. A diferencia del algoritmo clásico de PSO, este algoritmo no utiliza la mejor posición global para actualizar la velocidad de las partículas, sino que utiliza el conjunto de las mejores posiciones históricas de cada partícula, con el fin de no estancarse en un óptimo local, y mejorar la capacidad de búsqueda del algoritmo. Para realizar los experimentos, se utilizó el conjunto de problemas de prueba del CEC 2008 [61] con 1000 dimensiones.

En general estos métodos proponen el uso de la estrategia "divide y vencerás", o bien el uso de poblaciones grandes para resolver los problemas de alta dimensionalidad. Sin embargo, el uso de poblaciones grandes implica un consumo mayor de recursos de cómputo para poder realizar la búsqueda. Adicionalmente, el tiempo que se requiere para subdividir un problema es mucho mayor. En la propuesta que se presenta en el siguiente capítulo, se emplea un algoritmo que utiliza micro-población, que tiene como máximo cinco individuos, para realizar la búsqueda. Además, los problemas no se dividen en subproblemas, sino que se resuelven directamente, lo cual reduce el tiempo de ejecución del algoritmo, al evitar este procedimiento. Y para evitar el problema de la pérdida de diversidad, se emplea una reinicialización, que a diferencia de las técnicas tradicionales que proponen un reinicio aleatorio, incorpora el uso de una búsqueda local.

Capítulo 5

Algoritmo micro-poblacional basado en cúmulos de partículas para problemas de optimización de alta dimensionalidad

La idea principal del algoritmo propuesto en esta tesis es mantener un cúmulo de partículas, usando la micro-población propuesta en el algoritmo 5, de tal forma que se mejore la capacidad de búsqueda en problemas de alta dimensionalidad. Como se mencionó en el capítulo anterior, la complejidad de los problemas de alta dimensionalidad es elevada, lo que conlleva a mejorar la capacidad de búsqueda, sin que esto tenga un impacto negativo en el tiempo requerido para ello. Por tal motivo se propone, para aprovechar el uso de los demás operadores presentes, descartar el uso de la mejor posición de cada partícula para actualizar la velocidad, como se hace en el algoritmo básico de PSO (ver algoritmo 4). En su lugar, se modifica el método de reinicialización utilizado en el algoritmo 6, por una búsqueda local utilizando una función de distribución Gaussiana, con lo que se pretende utilizar el proceso de reinicialización de una forma explotativa en lugar de una forma explorativa. Adicionalmente, se propone agregar un valor de inercia [26] variable, en lugar de usar un valor constante, para actualizar la velocidad. También se incluye un control de velocidad, utilizando como velocidades máximas los límites del espacio de búsqueda, y se implementa un operador de mutación, con el cual se mejora la capacidad de búsqueda del algoritmo.

5.1. Descripción del algoritmo

La notación utilizada para describir el algoritmo 6, se presenta a continuación:

- N: Es el número de partículas.
- D: Es el número de dimensiones que tiene el problema

- UB: Es el límite superior del dominio del espacio de búsqueda.
- LB: Es el límite inferior del dominio del espacio de búsqueda.
- NR: Es el número de soluciones a reemplazarse utilizando la búsqueda local.
- GR: Es el número de generaciones de reemplazo, es decir, el número de generaciones que deberán transcurrir, sin encontrar una mejor solución, antes de aplicar la búsqueda local.
- $contGR$: Contador de generaciones de reemplazo, el cual es incrementado por cada iteración donde no se encuentra una mejor solución global. Este contador se reinicializa si se encuentra una mejor solución global.
- Gen_{Act}: Es el número que denota la generación actual, es decir, la iteración en la que se encuentra el proceso de búsqueda.
- Gen_{Max}: Es el número máximo de iteraciones que realizará el algoritmo antes de reportar la mejor solución encontrada.
- x_{id}: Posición actual de la partícula i en la dimensión d.
- v_{id}: Velocidad de la partícula i en la dimensión d, con la cual se actualizará la posición de x_{id}.
- c_1: Es un valor constante, utilizado para actualizar v_{id}.
- r_1: Es un valor aleatorio, entre 0 y 1, utilizado para actualizar v_{id}.
- ω: Valor de la inercia, empleado para actualizar v_{id}.
- PM: Es la probabilidad de mutación.

Los parámetros de entrada que tiene el algoritmo son: el número de partículas, el número de soluciones que se van a reemplazar, el número de generaciones que deberán transcurrir antes de realizar una búsqueda local con el operador de reinicio, la constante c_1, la probabilidad de mutación PM, y una semilla para el generador de números aleatorios, que consiste de un número real en el intervalo de 0 a 1.

5.1.1. Operador de búsqueda local

El uso de poblaciones de tamaño pequeño acelera la pérdida de diversidad en cada iteración, lo cual hace que sea una práctica poco común el uso de poblaciones muy pequeñas [62]. Sin embargo, desde un punto de vista teórico, es factible utilizar poblaciones muy pequeñas (no mayores a 5 individuos) si se implementan apropiadamente procesos de reinicialización para compensar el problema de pérdida de diversidad [63]. En esta propuesta, en lugar de utilizar formalmente un operador de reinicialización (el cual actualiza la velocidad y la posición de cada partícula reiniciada con valores

Algoritmo 6 Pseudocódigo del algoritmo propuesto

Entrada: N, NR, GR, c_1, $PM \in [0,1]$, semilla para generar números aleatorios $\in [0,1]$

Salida: La mejor solución encontrada

```
1: Inicializar aleatoriamente las posiciones y velocidades de las partículas
2: for Gen_Act = 1 to Gen_Max do
3:     if ContGR = GR then
4:         Aplicar operador de búsqueda local
5:         contGR = 0
6:     end if
7:     Encontrar g_Best
8:     for i = 0 to N do
9:         for d = 0 to D do
10:            Actualizar la inercia ω
11:            Actualizar la velocidad v_id
12:            Actualizar la posición x_id
13:        end for
14:    end for
15:    Aplicar operador de mutación
16: end for
17: Reportar la mejor solución encontrada.
```

aleatorios), se propone utilizar una búsqueda local, utilizando una función de distribución Gaussiana, para modificar la posición de las partículas. Para ello, se ordenan todas las partículas con base en su mejor posición. Posteriormente, se reinicializa la velocidad, con un valor aleatorio entre 0 y 1. Tomando como referencia [64], se propone modificar la posición actual de la partícula utilizando:

$$x_{id} := N(\mu, \sigma) \tag{5.1}$$

Donde N denota que se usará una distribución Gaussiana. Tanto μ como σ están dados en esta propuesta por:

$$\mu = \frac{gbest - x_{id}}{2.0} \tag{5.2}$$

$$\sigma = \sqrt{\mid gbest - x_{id} \mid} \tag{5.3}$$

Este proceso permite que se realice una búsqueda, donde existe una mayor probabilidad de actualizar la posición en dirección a la mejor posición global, con un rango de búsqueda delimitado por $\sqrt{\mid gbest - x_{id} \mid}$, tal y como se muestra en la figura 5.1. Esta modificación respecto al proceso tradicional de reinicialización, se plantea con el propósito de reducir la cantidad de operaciones realizadas por cada iteración. Es decir, en lugar de reiniciar las posiciones de cada partícula aleatoriamente, se busca mejorar la capacidad explotativa con el reinicio, para que la búsqueda realizada en

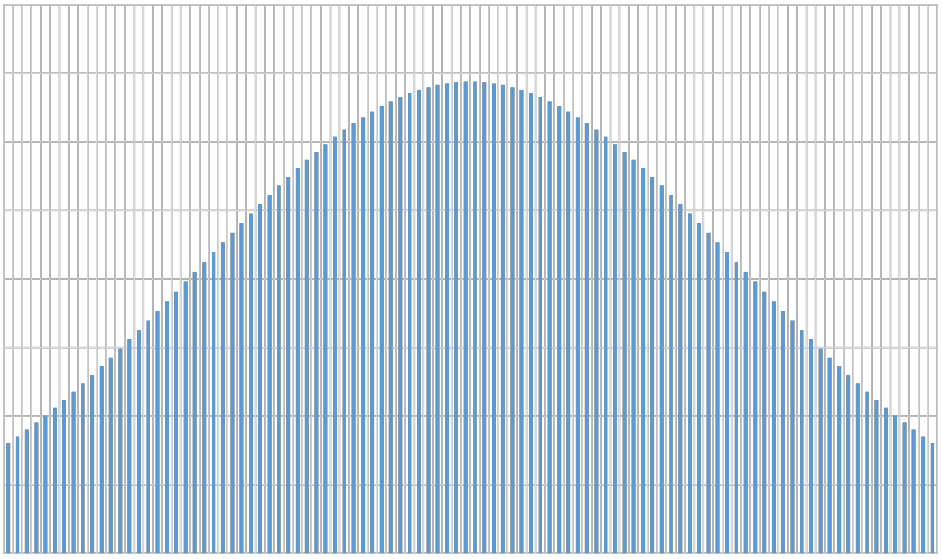

Figura 5.1: Representación gráfica de la campana de Gauss

cada iteración por medio de la aceleración y la mutación, tengan mejores capacidades exploratívas. El algoritmo 7 muestra la función de búsqueda local implementado en esta propuesta.

Algoritmo 7 Función de búsqueda local

Entrada: Número de partículas, Número de soluciones para reemplazar NR, generación de reemplazo GR

Salida: La mejor solución encontrada

Ordenar las partículas, con base en su mejor posición, de menor a mayor
for i = Número de partículas - NR to Número de partículas **do**
  for $j = 0$ to *NumDimensiones* **do**
    $v_{id} := rand()$; //Número aleatorio entre 0 y 1
    $\mu := \frac{gbest - x_{id}}{2.0}$
    $\sigma := \sqrt{\mid gbest - x_{id} \mid}$
    $x_{id} := N(\mu, \sigma)$ // Número aleatorio con distribución Gaussiana
  end for
end for

5.1.2. Búsqueda de g_{best}

Dentro de este proceso de búsqueda del óptimo global g_{best}, implícitamente se realiza la búsqueda de las mejores posiciones locales (p_{best}), las cuales se comparan entre sí para encontrar, con base en la función objetivo, a la que tiene la mejor posición entre las partículas. Adicionalmente, se utilizan como variables auxiliares $bestfx_i$ que almacena el mejor valor de $f(x)$ obtenido históricamente por la partícula, cuyas posiciones $\vec{x}$ están denotadas por p_{best}. Este proceso de búsqueda se detalla en el algoritmo 8.

5.1.3. Actualización de la inercia

Como se mencionó en el capítulo 3, el factor de inercia modifica la velocidad, con el objetivo de no sobrepasar los límites del espacio de búsqueda. Para esta propuesta, se plantea que el valor de la inercia sea incrementado desde 0 hasta 0.5, con el objetivo de que el impacto de la inercia sea mayor conforme se vaya iterando. Esto con el propósito de dar más libertad a la búsqueda al inicio del algoritmo. El valor de la inercia está dado por:

$$\omega = 0.5 * \left(\frac{Gen_{Act}}{Gen_{Max}} \right) \tag{5.4}$$

En casos de que la velocidad excediera los límites del espacio de búsqueda, ésta será limitada en el proceso de actualización de velocidad, como se presenta en la siguiente sección.

5.1.4. Actualización de la velocidad

Para actualizar la velocidad, como se mencionó al inicio de este capítulo, se realiza una modificación a la función original propuesta en [24], que está dada por:

$$\vec{v}_i(t+1) \leftarrow \vec{v}_i(t) + c_1 \cdot r_1 \cdot (p_{Best} - \vec{x}_i) + c_2 \cdot r_2 \cdot (g_{Best} - \vec{x}_i) \tag{5.5}$$

Para su implementación en la propuesta presentada en esta tesis, a la fórmula de aceleración le es eliminado el parámetro de la mejor posición local p_{Best}, debido a que la capacidad explotativa del algoritmo recae en el operador de búsqueda local mencionado anteriormente, y en su lugar sólo se deja la aceleración utilizando la mejor posición global como punto de referencia para la búsqueda. Adicionalmente, se agrega el factor de inercia para controlar la velocidad. Con ello, la fórmula de aceleración queda dada por:

$$\vec{v}_i(t+1) = \omega * \vec{v}_i(t) + c_1 * r_1 * (g_{best} - \vec{x}_i) \tag{5.6}$$

Donde ω es el factor de inercia explicado en la sección anterior, c_1 es un número real constante, r_1 es un número real generado aleatoriamente, en el intervalo de 0 a 1, $\vec{v}_i(t)$ es la velocidad actual asociada a la partícula i, $\vec{x}_i$ es la solución dada por la partícula i.

Algoritmo 8 Función de búsqueda de g_{best}

for $i = 0$ to N **do**
  Evaluar con base en la función objetivo
  Seleccionar la mejor posición local de la partícula:
  if $Gen_{Act} = 0$ **then**
    $bestfx_i = fx_i$
    $contGR := 0$
  else
    if $fx_i \leq bestfx_i$ **then**
      $bestfx_i = fx_i$
      $p_{best} = \vec{x}_i$
    end if
  end if
  elegir a la mejor partícula global:
  if $i = 0$ **then**
    $best = i$
  else
    if $bestfx_i \leq bestfx_{best}$ **then**
      $best = i$
    end if
  end if
  if $Gen_{Act} = 0$ **then**
    $g_{best} = best$
    $contGR := 0$
  else
    if $bestfx_i \leq bestfx_{best}$ **then**
      $bestfx_i = fx_i$
      $g_{best} = \vec{x}_i$
      $contGR := 0$
    else
      $contGR := contGR + 1$
    end if
  end if
end for

Para evitar que la velocidad exceda los límites del espacio de búsqueda, además del factor de inercia, se utiliza un control de velocidad, donde las velocidades máximas están dadas por los límites superior e inferior del espacio de búsqueda, quedando de la siguiente manera:

$$\vec{v}_{i,j} = \begin{cases} \vec{v}_{i,j} & \text{si } \vec{v}_{i,j} \leq UB \text{ y } \vec{v}_{i,j} \geq LB \\ UB & \text{si } \vec{v}_{i,j} > UB \\ LB & \text{si } \vec{v}_{i,j} < LB \end{cases} \tag{5.7}$$

Una vez que se tiene la velocidad actualizada, se procede a actualizar la posición de la partícula como se describe a continuación:

$$\vec{x}_i = \vec{x}_i + \vec{v}_i \tag{5.8}$$

5.1.5. Operador de mutación

En su versión original, el algoritmo de PSO no posee un operador de mutación; sin embargo, éste puede ser incluido para mejorar la capacidad de búsqueda del PSO al ser utilizado como optimizador. El uso del operador de mutación es relativamente común en las técnicas de computación evolutiva, debido a que previene la pérdida de diversidad en las soluciones, lo cual implica tener una mayor cobertura del espacio de soluciones. El operador de mutación crea una variación del individuo al que se le aplica el operador, la cual puede prevenir que la búsqueda se estanque en un óptimo local [65].

Para el algoritmo propuesto, se implementó un operador de mutación desarrollado por Michalewicz originalmente para algoritmos genéticos [66]. Para el funcionamiento de este operador, se realiza una suma o una resta a la posición actual de la partícula, en todas las dimensiones, cuya magnitud depende del número de iteración actual. Con esto se permite realizar una variación grande cuando se comienza la búsqueda, y conforme vaya avanzando la búsqueda, los cambios que se realicen son menores. La definición de la función aplicada es la siguiente:

$$x_{i,d} = \begin{cases} x_{i,d} + \Delta(t, UB - x_{id}) & \text{si } R = 0 \\ x_{i,d} - \Delta(t, x_{id} - LB) & \text{si } R = 1 \end{cases} \tag{5.9}$$

Donde t es el número de iteración actual, UB es el límite superior del dominio del espacio de búsqueda, LB es el límite inferior del dominio del espacio de búsqueda, R es un bit generado aleatoriamente, donde 0 y 1 tienen 50 % de probabilidad de ser generados, y $\Delta(t, y)$ es una función que entrega un valor en el intervalo $[0, y]$. La función $\Delta(t, y)$ está definida por:

$$\Delta(t, y) = y * \left(1 - r^{1-\left(\frac{t}{T}\right)^b}\right) \tag{5.10}$$

Donde r es un número aleatorio generado de una distribución uniforme en el intervalo $[0, 1]$, T es el máximo número de iteraciones y b es un parámetro que define el nivel de no-uniformidad del operador. Para este algoritmo, se asigna $b = 5$, como se sugiere en [66].

5.2. Estudio experimental

Para validar la propuesta realizada en esta tesis, se llevó a cabo un estudio experimental, donde se evalúa un conjunto de problemas de prueba utilizando el algoritmo propuesto, dados ciertos parámetros de entrada. Posteriormente se comparan las soluciones dadas con respecto a los valores óptimos y los resultados de otros algoritmos.

5.2.1. Problemas de prueba

Para comparar el desempeño del algoritmo propuesto con respecto a otras metaheurísticas, se decidió utilizar el conjunto de pruebas del CEC 2008 propuestos por Tang et al. [61], el cual es muy utilizado para validar problemas mono-objetivo de alta dimensionalidad, especialmente porque pueden ser utilizados para probar distintas dimensionalidades, llegándose hasta las 1000 dimensiones. En la competencia se propone realizar un máximo de $5.00E+03 \times$ `número de dimensiones` de evaluaciones de la función objetivo. En la tabla 5.1 se muestran las características de los problemas utilizados de forma general.
Los problemas de prueba que se utilizaron, se detallan a continuación.

Función	Problema	Separabilidad	Modalidad	Escalabilidad
f_1	Función desplazada de la esfera	separable	unimodal	escalable
f_2	Función desplazada del problema de Schwefel 2.21	no separable	unimodal	escalable
f_3	Función desplazada de Rosenbrock	no separable	multimodal	escalable
f_4	Función desplazada de Rastrigin	separable	multimodal	escalable
f_5	Función desplazada de Griewank	no separable	multimodal	escalable
f_6	Función desplazada de Ackley	separable	multimodal	escalable

Tabla 5.1: Características de los problemas de prueba usados para la competencia de optimización a gran escala realizada durante el CEC 2008

f_1 función desplazada de la esfera

$$F_1(x) = \sum_{i=1}^{D} z_i^2 + f_{bias_1}, \mathbf{z} = \mathbf{x} - \mathbf{o}, \mathbf{x} = [\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_D}] \tag{5.11}$$

Donde D es el número de dimensiones, $o = [o_1, o_2, \cdots, o_D]$ es el óptimo global desplazado.
$x \in [-100, 100]^D$, el óptimo global: x* = o, $F_1(x*) = f_{bias_1} = -450$

f_2 Función desplazada del problema de Schwefel 2.21

$$F_2(x) = max_i \| z_i \|, 1 \leq i \leq D + f_{bias_2}, \mathbf{z} = \mathbf{x} - \mathbf{o}, \mathbf{x} = [\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_D}] \quad (5.12)$$

Donde D es el número de dimensiones, $o = [o_1, o_2, \cdots, o_D]$ es el óptimo global desplazado.
$x \in [-100, 100]^D$, el óptimo global: x* = o, $F_2(x*) = f_{bias_2} = -450$

f_3 Función desplazada de Rosenbrock

$$F_3(x) = \sum_{i=1}^{D-1} \left(100(z_i^2 - z_{i-1})^2 + (z_i - 1)^2\right) + f_{bias_3}, \mathbf{z} = \mathbf{x} - \mathbf{o}, \mathbf{x} = [\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_D}] \quad (5.13)$$

Donde D es el número de dimensiones, $o = [o_1, o_2, \cdots, o_D]$ es el óptimo global desplazado.
$x \in [-100, 100]^D$, el óptimo global: x* = o, $F_3(x*) = f_{bias_3} = 390$

f_4 Función desplazada de Rastrigin

$$F_4(x) = \sum_{i=1}^{D} (z_i^2 - 10cos(2\pi z_i) + 10) + f_{bias_4}, \mathbf{z} = \mathbf{x} - \mathbf{o}, \mathbf{x} = [\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_D}] \quad (5.14)$$

Donde D es el número de dimensiones, $o = [o_1, o_2, \cdots, o_D]$ es el óptimo global desplazado.
$x \in [-5, 5]^D$, el óptimo global: x* = o, $F_4(x*) = f_{bias_4} = -330$

f_5 Función desplazada de Griewank

$$F_5(x) = \sum_{i=1}^{D} \frac{z_i^2}{4000} - \prod_{i=1}^{D} cos\left(\frac{z_i}{\sqrt{i}}\right) + 1 + f_{bias_5}, \mathbf{z} = \mathbf{x} - \mathbf{o}, \mathbf{x} = [\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_D}] \quad (5.15)$$

Donde D es el número de dimensiones, $o = [o_1, o_2, \cdots, o_D]$ es el óptimo global desplazado.
$x \in [-600, 600]^D$, el óptimo global: x* = o, $F_5(x*) = f_{bias_5} = -180$

f_6 Función desplazada de Ackley

$$F_6(x) = -20exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2}\right) - exp\left(\frac{1}{D}\sum_{i=1}^{D} cos(2\pi z_i)\right) + 20 + e + f_{bias_6} \quad (5.16)$$

$\mathbf{z} = \mathbf{x} - \mathbf{o}, \mathbf{x} = [\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_D}]$.
Donde D es el número de dimensiones, $o = [o_1, o_2, \cdots, o_D]$ es el óptimo global desplazado.
$x \in [-32, 32]^D$, el óptimo global: x* = o, $F_6(x*) = f_{bias_6} = -140$

5.2.2. Metodología empleada

Se realizaron 30 ejecuciones independientes por cada problema de prueba, con 100 y 1000 dimensiones. El número de evaluaciones de la función objetivo quedó limitado a $5.00E+03 \times$ `número de dimensiones`, utilizando los parámetros descritos en la sección anterior. Los resultados se compararon con los descritos en los artículos mencionados en el estado del arte. Las ejecuciones fueron realizadas en un equipo de cómputo con un procesador AMD A6, con una frecuencia de reloj de 2.7 GHz y 6 GB de memoria RAM.

5.2.3. Análisis de varianza

Se realizó un análisis de varianza (ANOVA) para determinar la sensibilidad del algoritmo a sus parámetros. Para ello, primero se modificó la función de actualización de velocidad descrito en la ecuación 5.6, utilizando:

$$\vec{v}_i(t+1) = \omega * \vec{v}_i(t) + c_1 * r_1 * (g_{best} - \vec{x}_i) + c_2 * r_1 * (p_{best} - \vec{x}_i) \quad (5.17)$$

donde c_2 es una constante y r_2 es un número aleatorio en el intervalo $[0, 1]$. Los parámetros analizados fueron GR, c_1 y c_2 para 100, 500 y 1000 dimensiones.
Los valores para cada parámetro se definieron de la siguiente manera:

- GR: 25, 50, 100.
- c_1: 0, 0.5, 1.0, 1.7
- c_2: 0, 0.5, 1.0, 1.7

En total se probaron 71 combinaciones de parámetros, donde la única combinación no realizada, fue aquella donde c_1 y c_2 son iguales a 0. Para cada combinación se realizaron 30 ejecuciones distintas por cada función de prueba y por cada dimensión. En total se realizaron 38,340 ejecuciones independientes.
Las hipótesis utilizadas en este experimento fueron las siguientes:

- Hipótesis nula: No existe diferencia significativa entre los promedios de los resultados obtenidos, y si existen diferencias están dadas por efectos de aleatoriedad.
- Hipótesis alternativa: Existe una combinación de factores donde los promedios son diferentes y no están dados por efectos de aleatoriedad.

Con base en los resultados obtenidos, se comprobó que se cumple la hipótesis nula en la mayoría de las combinaciones. No obstante, en las funciones f_4 y f_5 se cumplió la hipótesis alternativa para las tres dimensiones, cuyos resultados mostraron lo siguiente:

- Para la función f_4 con 100 dimensiones: Los mejores resultados se obtuvieron con $GR = 25$, $c_1 = 0.5$, $c_2 = 0$
- Para la función f_4 con 500 dimensiones: Los mejores resultados se obtuvieron con $GR = 25$, $c_1 = 0.5$, $c_2 = 0$
- Para la función f_4 con 1000 dimensiones: Los mejores resultados se obtuvieron con $GR = 25$, $c_1 = 0.5$, $c_2 = 0$
- Para la función f_5 con 500 dimensiones: Los mejores resultados se obtuvieron con $GR = 100$, $c_1 = 1.7$, $c_2 = 0$
- Para la función f_5 con 1000 dimensiones: Los mejores resultados se obtuvieron con $GR = 100$, $c_1 = 1.7$, $c_2 = 0$

Con los resultados anteriores se pudo confirmar la selección de parámetros utilizada en la experimentación con el algoritmo propuesto. Adicionalmente se comprobó que la constante c_2 que acelera la velocidad con base a la mejor posición local, tiene una influencia prácticamente nula en la aceleración de las partículas, lo que permite que se pueda eliminar de la fórmula, conservando la ecuación (5.6) como función de aceleración. Con respecto a GR, se observó que su influencia fue mayor en la función f_4, donde el resultado mejora si es menor el valor de GR. No obstante, en la función f_5 tuvo un mejor desempeño el algoritmo cuando GR tuvo un valor mayor, sin que tuviera una diferencia significativa con respecto a $GR = 50$. Por lo tanto se puede adoptar $GR = 50$. Finalmente, el valor de c_1 en general tuvo un mejor desempeño cuando se adoptó un valor elevado, siendo adecuado el uso de $c_2 = 1.7$.

5.2.4. Parámetros utilizados

Para realizar las pruebas, se tuvieron que ajustar los parámetros de entrada del algoritmo propuesto, con el fin de adaptarse al máximo número de evaluaciones de la función objetivo, conservando el objetivo inicial de mantener una población pequeña. Los parámetros de entrada para cada una de las pruebas son:

- Tamaño de población: 4

- Número de iteraciones: $1250 \times$ `número de dimensiones`
- $c_1 = 1.7$
- Número de iteraciones sin mejora previo al reemplazo: 50
- Número de partículas a ser reemplazadas: 3
- Porcentaje de mutación $= \frac{1}{\texttt{número de dimensiones}}$

Los parámetros empleados por los algoritmos con los que se comparó el algoritmo propuesto se presentan en la tabla 5.2.

	CSO	CCPSO2	MLCC	SEP-CMA-ES	EPUS-PSO	DMS-PSO
Tamaño población	25-300	30	50	240	20	450
Factor social	0-0.3	-	-	-	-	-
Tamaño de grupo	200-1000	2-250	5-1000			2 - 5
SRS	-	-	-	-	0-0.5	
Inercia	-	-	-	-	-	0.9 - 0.2

Tabla 5.2: Parámetros usados para los problemas de prueba del CEC 2008

5.2.5. Análisis estadístico

Para poder medir la confiabilidad del algoritmo, es necesario realizar un análisis estadístico de las pruebas realizadas por cada función, y de esta manera, obtener los intervalos de confianza de la media estadística.
Una forma de obtener los intervalos de confianza de distribuciones desconocidas, es la **Distribución Muestral de Bootstrap** (DMB). Para obtener una DMB, se tienen que formar K muestras de tamaño N, donde cada muestra está formada por valores elegidos de manera aleatoria con reemplazo y probabilidad $\frac{1}{N}$ de ser elegido. Generalmente, se emplea $K = 1000$.
Para cada problema se obtuvo una DMB, y de cada una se obtuvieron sus intervalos de confianza con percentil 95 %. A continuación se presenta una tabla con los intervalos de confianza para cada función de prueba por cada tamaño de dimensión evaluada.

Observando la tabla 5.3, se puede observar que para la mayoría de las funciones se obtiene una aproximación cercana al óptimo, donde los intervalos son pequeños. Para las funciones f_1, f_2, f_4 y f_6 se tienen los intervalos más pequeños, donde la variación apenas se da por decimales. En la función f_5 se observa que se tuvo una mejor aproximación al óptimo con 500 dimensiones, teniendo pocos valores con menor precisión de decimales.

función/intervalos de confianza	100 dim	500 dim	1000 dim
f_1	$[6.71E-13, 7.48E-13]$	$[3.73E-12, 3.89E-12]$	$[7.53E-12, 7.84E-12]$
f_2	$[1.07E-03, 1.24E-03]$	$[3.40E-01, 3.63E-01]$	$[3.49E+00, 3.62E+00]$
f_3	$[3.95E+02, 1.25E+03]$	$[8.88E+02, 1.12E+03]$	$[1.73E+03, 1.83E+03]$
f_4	$[1.06E+00, 1.79E+00]$	$[9.82E+00, 1.23E+01]$	$[2.19E+01, 2.54E+01]$
f_5	$[3.86E-03, 9.84E-03]$	$[1.80E-12, 1.72E-03]$	$[7.39E-04, 3.61E-03]$
f_6	$[3.33E-10, 3.83E-10]$	$[4.08E-10, 4.42E-10]$	$[4.21E-10, 4.58E-10]$

Tabla 5.3: Intervalos de confianza para 100, 500 y 1000 dimensiones para las 6 funciones de prueba

5.2.6. Comparación de resultados

Una vez que se realizaron las ejecuciones del algoritmo, se calculó el la diferencia entre el resultado obtenido y el óptimo global de cada problema. Posteriormente se calculó el promedio y la desviación estándar de las diferencias por cada problema, las cuales se compararon con los resultados obtenidos en los problemas del estado del arte. Estas comparaciones se ven reflejadas en las tablas que se muestran a continuación.

función		LS-μ-PSO	CSO	CCPSO2	MLCC	sep-CMA-ES	EPUS-PSO	DMS-PSO
f_1	Promedio	7.07E-13	9.11E-29	7.73E-14	6.82E-14	9.02E-15	7.47E-01	**0.00E+00**
	Desv est.	1.29E-13	1.10E-28	3.23E-14	2.32E-14	5.53E-15	1.70E-01	**0.00E+00**
f_2	Promedio	**1.16E-03**	3.35E+01	6.08E+00	2.53E+01	2.31E+01	1.86E+01	3.65E+00
	Desv est.	**2.82E-04**	5.38E+00	7.83E+00	8.73E+00	1.39E+01	2.26E+00	7.30E-01
f_3	Promedio	7.92E+02	3.90E+02	4.23E+02	1.50E+02	**4.31E+00**	4.99E+03	2.83E+02
	Desv est.	1.44E+03	5.53E+02	8.65E+02	5.72E+01	**1.26E+01**	5.35E+03	9.40E+02
f_4	Promedio	1.43E+00	5.60E+01	3.98E-02	**4.39E-13**	2.78E+02	4.71E+02	1.83E+02
	Desv est.	1.17E+00	7.48E+00	1.99E-01	**9.21E-14**	3.43E+01	5.94E+01	2.16E+01
f_5	Promedio	3.66E-13	**0.00E+00**	3.45E-03	3.41E-14	2.96E-04	3.72E-01	**0.00E+00**
	Desv est.	5.42E-14	**0.00E+00**	4.88E-03	1.16E-14	1.48E-03	5.60E-02	**0.00E+00**
f_6	Promedio	3.58E-10	1.20E-14	1.44E-13	1.11E-13	2.12E+01	2.06E+00	**0.00E+00**
	Desv est.	8.34E-11	1.52E-15	3.06E-14	7.87E-15	4.02E-01	4.40E-01	**0.00E+00**

Tabla 5.4: Comparación del margen de error entre los resultados obtenidos y el óptimo global, del algoritmo propuesto con respecto a algoritmos del estado del arte en problemas con 100 dimensiones

Observando las comparaciones realizadas en ambas tablas, se puede notar que el rendimiento de esta propuesta es competitiva en la resolución de problemas de alta dimensionalidad, teniendo un desempeño destacado en la función f_2 al superar a los demás algoritmos en los resultados obtenidos. Otro punto a destacar, es que el cambio de dimensionalidad no empeora en gran medida el desempeño del algoritmo, comparando los resultados para el mismo problema. Esto es destacado como una característica principal del algoritmo propuesto, lo cual le permite adaptarse a la escalabilidad de un problema dado.
Por otro lado, el desempeño del algoritmo se ve afectado en gran medida con la función f_3, donde si bien no tiene un resultado muy distante respecto al obtenido por

función		LS-μ-PSO	CSO	CCPSO2	MLCC	sep-CMA-ES	EPUS-PSO	DMS-PSO
f_1	Promedio	3.81E-12	6.57E-23	7.73E-14	4.30E-13	2.25E-14	8.45E+01	**0.00E+00**
	Desv est.	2.66E-13	3.90E-24	3.23E-14	3.31E-14	6.10E-15	6.40E+00	**0.00E+00**
f_2	Promedio	**3.52E-01**	2.60E+01	5.79E+01	6.67E+01	2.12E+02	4.35E+01	6.89E+01
	Desv est.	**3.84E-02**	2.40E+00	4.21E+01	5.70E+00	1.74E+01	5.51E-01	2.01E+00
f_3	Promedio	9.89E+02	5.74E+02	7.24E+02	9.25E+02	**2.93E+02**	5.77E+04	4.67E+07
	Desv est.	3.77E+02	1.67E+02	1.54E+02	1.73E+02	**3.59E+01**	8.04E+03	5.87E+06
f_4	Promedio	1.10E+01	3.19E+02	3.98E-02	**1.79E-11**	2.18E+03	3.49E+03	1.61E+03
	Desv est.	1.17E+00	2.16E+01	1.99E-01	**6.31E-11**	1.51E+02	1.12E+02	1.04E+02
f_5	Promedio	7.39E-04	2.22E-16	1.18E-03	2.13E-13	7.88E-04	1.64E+00	**0.00E+00**
	Desv est.	2.93E-03	0.00E+00	4.61E-03	2.48E-14	2.82E-03	4.69E-02	**0.00E+00**
f_6	Promedio	4.25E-10	4.13E-13	5.34E-13	**5.34E-13**	2.15E+01	6.64E+00	2.00E+00
	Desv est.	5.71E-11	1.10E-14	8.61E-14	**7.01E-14**	3.10E-01	4.49E-01	9.66E-02

Tabla 5.5: Comparación del margen de error entre los resultados obtenidos y el óptimo global, del algoritmo propuesto con respecto a algoritmos del estado del arte en problemas con 500 dimensiones

función		LS-μ-PSO	CSO	CCPSO2	MLCC	sep-CMA-ES	EPUS-PSO	DMS-PSO
f_1	Promedio	7.67E-12	1.09E-21	5.18E-13	8.46-13	7.81E-15	5.53E+02	**0.00E+00**
	Desv est.	5.18E-13	4.20E-23	9.61E-14	5.01E-14	1.52E-15	2.86E+01	**0.00E+00**
f_2	Promedio	**3.56E+00**	4.15E+01	7.82E+01	1.09E+02	3.65E+02	4.66E+01	9.15E+01
	Desv est.	**0.21E+00**	9.74E-01	4.25E+01	4.75E+00	9.02E+00	4.00E-01	7.14E-01
f_3	Promedio	1.78E+03	1.01E+03	1.33E+03	1.80E+03	**9.10E+02**	8.37E+05	8.98E+09
	Desv est.	1.45E+02	3.02E+01	2.63E+02	1.58E+02	**4.54E+01**	1.52E+05	4.39E+08
f_4	Promedio	2.28E+01	6.89E+02	1.99E-01	**1.37E-10**	5.31E+03	7.58E+03	3.84E+03
	Desv est.	5.50E+00	3.10E+01	4.06E-01	**3.37E-10**	2.48E+02	1.51E+02	1.71E+02
f_5	Promedio	3.62E-12	2.26E-16	1.18E-03	4.18E-13	3.94E-04	5.89E+00	**0.00E+00**
	Desv est.	2.16E-13	2.18E-17	3.27E-03	2.78E-14	1.97E-03	3.91E-01	**0.00E+00**
f_6	Promedio	4.38E-10	1.21E-12	**1.02E-12**	1.06E-12	2.15E+01	1.89E+01	7.76E+00
	Desv est.	6.59E-11	2.64E-14	**1.68E-13**	7.68E-14	3.19E-01	2.49E+00	8.92E-02

Tabla 5.6: Comparación del margen de error entre los resultados obtenidos y el óptimo global, del algoritmo propuesto con respecto a algoritmos del estado del arte en problemas con 1000 dimensiones

los algoritmos comparados, sí dista del óptimo global de manera considerable. Respecto a las demás funciones, se cuenta con un rendimiento competitivo, donde si bien no consigue ganar en la precisión del resultado, obtiene uno de los mejores resultados en su grupo de comparación. Si a esto se le agrega que el tiempo de ejecución de las pruebas fue considerablemente bajo, estando cercano a 20 segundos en 100 dimensiones y menor a 20 minutos para 1000 dimensiones, concluimos que el algoritmo tiene un buen desempeño para la resolución de problemas de alta dimensionalidad.

Capítulo 6

Conclusiones y trabajo a futuro

En este trabajo se propuso un μ-PSO para problemas de optimización mono-objetivo, el cual se llamó LS-μ-PSO. Se utilizó un mecanismo de búsqueda local para reemplazar la reinicialización habitualmente utilizada en algoritmos micropoblacionales. Adicionalmente se propuso evitar el uso de la mejor posición local ($Pbest$) en la aceleración de las partículas, como lo propone el algoritmo original de PSO.
El realizar la eliminación de la mejor posición local, permite que la búsqueda local en la reinicialización tenga mayor impacto, y el uso de una población pequeña permite reducir el consumo de memoria en la ejecución del algoritmo.
Los resultados obtenidos en las pruebas, muestran que LS-μ-PSO puede resolver problemas de alta dimensionalidad de forma satisfactoria, incluyendo problemas con hasta mil variables de decisión. A pesar de que no pudo superar en algunos problemas a ciertos algoritmos, como el DMS-PSO, se observó que su desempeño es competitivo, superando a la mayoría de los algoritmos usados en nuestro estudio comparativo en los problemas de prueba de la competencia del CEC 2008.

6.1. Trabajo a futuro

A pesar del desempeño aceptable que tuvo esta propuesta, existen varios aspectos que se pueden mejorar. Uno de ellos es la capacidad de búsqueda después de determinado número de iteraciones, donde si bien se siguen encontrando mejores soluciones, las mejoras no son significativas. Esto pudo deberse a la eliminación de la mejor posición local, aunque este cambio ayudó a aproximarse rápidamente a una buena solución. Este mecanismo deberá analizarse con más detenimiento, para determinar cómo adecuarlo para mejorar el desempeño de nuestro algoritmo.
Adicionalmente, se propone extender el uso de este algoritmo a problemas de optimización multi-objetivo, con el cual se puedan resolver problemas de alta dimensionalidad, en un tiempo razonable.

Bibliografía

[1] C. A. Coello Coello. Evolutionary multi-objective optimization: a historical view of the field. *IEEE Computational Intelligence Magazine*, 1(1):28–36, Feb 2006.

[2] Ke Tang, Xiaodong Li, P. N. Suganthan, Zhenyu Yang, and Thomas Weise. Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, 2009.

[3] J.C. Fuentes Cabrera. Un nuevo algoritmo de optimización basado en optimización mediante cúmulos de partículas utilizando tamaños de población muy pequeños. Master's thesis, Maestría en Ciencias en la Especialidad de Ingeniería Eléctrica: Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Marzo 2008.

[4] L.Y. Juárez. Un Estudio Empírico de Micro Algoritmos Inspirados en la Naturaleza en la Minimización de Inestabilidad Cíclica en Ambientes Inteligentes. Master's thesis, Maestría en Cómputo Aplicado: Laboratorio Nacional de Informática Avanzada (LANIA) A.C., Febrero 2015.

[5] P. Pedregal. *Introduction to optimization*. Springer, USA, 2003.

[6] L. C. W. Dixon. *Nonlineal optimization*. The English Universities Press, Glasgow, Great Britain,, 1973.

[7] S. S. Rao. *Optimization : theory and applications*. John Wiley and Sons, Ltd, New York, USA, 09 2001.

[8] A. Eiben and J. Smith. *Introduction To Evolutionary Computing*, volume 45. Springer, 2003.

[9] David B. Fogel. *An introduction to evolutionary computation and some applications*. John Wiley & Sons, 1999.

[10] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE transactions on neural networks*, 5 (1):3–14, 1994.

[11] A. Hoffman. *Arguments on evolution : a paleontologist's perspective / Antoni Hoffman*. Oxford University Press New York, 1989.

[12] E. Mayr. *Toward a New Philosophy of Biology: Observations of an Evolutionist*. Belknap Press of Harvard University Press, 1988.

[13] C. A. Coello Coello. An introduction to evolutionary algorithms and their applications. In *Proceedings of the 5th International Conference on Advanced Distributed Systems*, ISSADS'05, pages 425–442, Berlin, Heidelberg, 2005. Springer-Verlag.

[14] Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9, San Mateo, California, USA, 1991. Morgan Kaufmann Publishers.

[15] Lawrence J. Fogel. *Artificial Intelligence through Simulated Evolution*. John Wiley, 1966.

[16] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[17] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. volume 1 of *Foundations of Genetic Algorithms*, pages 69 – 93. Elsevier, 1991.

[18] B.P. Buckles and F.E. Petry. *Genetic Algorithms*. Technology Series. IEEE Computer Society Press, 1992.

[19] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5(1):96–101, Jan 1994.

[20] J. R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, Jun 1994.

[21] K. V. Price. New ideas in optimization. pages 79–108. McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.

[22] L. De Castro and J. Timmis. *Artificial immune systems: A new computational intelligence approach*. 06 2002.

[23] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, Feb 1996.

[24] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the 1995, IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, Nov 1995.

[25] James Kennedy, Russell C. Eberhart, and Yuhui Shi. chapter seven - the particle swarm. In *Swarm Intelligence*, The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann Publishers, San Francisco, California, USA, 2001.

[26] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pages 69–73, May 1998.

[27] Yuhui Shi and Russell C. Eberhart. Parameter selection in particle swarm optimization. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors, *Evolutionary Programming VII*, pages 591–600, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[28] 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, Feb 2002.

[29] Margarita Reyes-sierra and Carlos A. Coello Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *INTERNATIONAL JOURNAL OF COMPUTATIONAL INTELLIGENCE RESEARCH*, 2(3):287–308, 2006.

[30] S. Janson and M. Middendorf. A hierarchical particle swarm optimizer. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, volume 2, pages 770–776 Vol.2, Dec 2003.

[31] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation. (CEC'02) (Cat. No.02TH8600)*, volume 2, pages 1671–1676 vol.2, May 2002.

[32] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, Jun 2007.

[33] M. Lozano, D. Molina, and F. Herrera. Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. *Soft Computing*, 15(11):2085–2087, Nov 2011.

[34] Mohammad Nabi Omidvar, Xiaodong Li, and Ke Tang. Designing benchmark problems for large-scale continuous optimization. *Information Sciences*, 316:419 – 436, 2015.

[35] Daniel Molina Cabrera. Evolutionary algorithms for large-scale global optimisation: a snapshot, trends and challenges. *Progress in Artificial Intelligence*, 5(2):85–89, May 2016.

[36] Zhenyu Yang, Ke Tang, and Xin Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985 – 2999, 2008. Nature Inspired Problem-Solving.

[37] Zhenyu Yang, Ke Tang, and Xin Yao. Self-adaptive differential evolution with neighborhood search. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 1110–1116, June 2008.

[38] Zhenyu Yang, Ke Tang, and Xin Yao. Multilevel cooperative coevolution for large scale optimization. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 1663–1670, June 2008.

[39] M. N. Omidvar, X. Li, Z. Yang, and X. Yao. Cooperative co-evolution for large scale optimization through more frequent random grouping. In *IEEE Congress on Evolutionary Computation*, pages 1–8, July 2010.

[40] X. Li and X. Yao. Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms. In *2009 IEEE Congress on Evolutionary Computation*, pages 1546–1553, May 2009.

[41] X. Li and X. Yao. Cooperatively coevolving particle swarms for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 16(2):210–224, April 2012.

[42] Yuhua Li, Zhi-Hui Zhan, Shujin Lin, Jun Zhang, and Xiaonan Luo. Competitive and cooperative particle swarm optimization with information sharing mechanism for global optimization problems. *Information Sciences*, 293:370 – 382, 2015.

[43] Xin Yao, Yong Liu, and Guangming Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102, Jul 1999.

[44] C. Sun, Y. Jin, R. Cheng, J. Ding, and J. Zeng. Surrogate-assisted cooperative swarm optimization of high-dimensional expensive problems. *IEEE Transactions on Evolutionary Computation*, 21(4):644–660, Aug 2017.

[45] M. Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, page 1957 Vol. 3, 1999.

[46] Ran Cheng and Yaochu Jin. A social learning particle swarm optimization algorithm for scalable optimization. *Information Sciences*, 291(C):43–60, January 2015.

[47] Saúl Zapotecas Martínez and Carlos A. Coello Coello. MOEA/D assisted by RBF networks for expensive multiobjective optimization problems. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 1405–1412, New York, NY, USA, 2013. ACM.

[48] B. Liu, Q. Zhang, and G. G. E. Gielen. A gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems. *IEEE Transactions on Evolutionary Computation*, 18(2):180–192, April 2014.

[49] D. Lim, Y. Jin, Y. S. Ong, and B. Sendhoff. Generalizing surrogate-assisted evolutionary computation. *IEEE Transactions on Evolutionary Computation*, 14(3):329–355, June 2010.

[50] Can Liu and Bin Li. Individual-based cooperative coevolution local search for large scale optimization. In Hisashi Handa, Hisao Ishibuchi, Yew-Soon Ong, and Kay Chen Tan, editors, *Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Volume 1*, pages 535–547, Cham, 2015. Springer International Publishing.

[51] Francisco J. Solis and Roger J. B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, February 1981.

[52] Xiaodong Li, Ke Tang, Mohammmad Nabi Omidvar, Zhenyu Yang, and Kai Qin. Benchmark functions for the CEC'2013 special session and competition on large-scale global optimization. Technical report, Evolutionary Computationand Machine Learning Group, RMIT University, Australia, 2013.

[53] R. Cheng and Y. Jin. A competitive swarm optimizer for large scale optimization. *IEEE Transactions on Cybernetics*, 45(2):191–204, Feb 2015.

[54] S. T. Hsieh, T. Y. Sun, C. C. Liu, and S. J. Tsai. Efficient population utilization strategy for particle swarm optimizer. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):444–456, April 2009.

[55] 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 *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 3845–3852, June 2008.

[56] Sun Liang, Shinichi Yoshida, and Liang Yanchun. Cooperative particle swarm optimization for large scale numerical optimization. *Soft Computing and Intelligent Systems and International Symposium on Advanced Intelligent Systems*, 2010:892–897, 2010.

[57] C. Seren. A hybrid jumping particle swarm optimization method for high dimensional unconstrained discrete problems. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 1649–1656, June 2011.

[58] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers Operations Research*, 24(11):1097–1100, November 1997.

[59] K. E. Parsopoulos and M. N. Vrahatis. On the computation of all global minimizers through particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):211–224, June 2004.

[60] W. Guo, C. Si, Y. Xue, Y. Mao, L. Wang, and Q. Wu. A grouping particle swarm optimizer with personal-best-position guidance for large scale optimization. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 1–1, 2018.

[61] Ke Tang, Xiaodong Li, Ponnuthurai Suganthan, Zhenyu Yang, and Thomas Weise. Benchmark functions for the CEC'2008 special session and competition on large scale global optimization. Technical report, 12 2009.

[62] Juan C. Fuentes Cabrera and Carlos A. Coello Coello. Handling constraints in particle swarm optimization using a small population size. In Alexander Gelbukh and Ángel Fernando Kuri Morales, editors, *MICAI 2007: Advances in Artificial Intelligence*, pages 41–51, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[63] Carlos A. Coello Coello Coello and Gregorio Toscano Pulido. A micro-genetic algorithm for multiobjective optimization. In Eckart Zitzler, Lothar Thiele, Kalyanmoy Deb, Carlos Artemio Coello Coello, and David Corne, editors, *Evolutionary Multi-Criterion Optimization*, pages 126–140, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[64] R. A. Krohling and E. Mendel. Bare bones particle swarm optimization with gaussian or cauchy jumps. In *2009 IEEE Congress on Evolutionary Computation*, pages 3285–3291, May 2009.

[65] P. S. Andrews. An investigation into mutation operators for particle swarm optimization. In *2006 IEEE International Conference on Evolutionary Computation*, pages 1044–1051, 2006.

[66] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)*. Springer-Verlag, Berlin, Heidelberg, 1996.