Universidad de las Ciencias Informáticas

Facultad 6

Título: Biblioteca de Optimización por Enjambre de Partículas sobre la Plataforma T-arenal.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor:

Ernesto Contreras Torres

Tutores:

MSc. Dannier Trinchet Almaguer

Ing. César Raúl García Jacas

La Habana**, Junio, 2012**

"Año 54 de la Revolución"

Declaración de Autoría

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Ernesto Contreras Torres **MSc.** Dannier Trinchet Almaguer **Ing.** César Raúl García Jacas

_____ _____ _____

Firma del Autor Firma Tutor Firma del Tutor

Datos de Contacto

Autor:

Ernesto Contreras Torres

Universidad de las Ciencias Informáticas

Email: econtreras@estudiantes.uci.cu

Tutor 1:

Dannier Trinchet Almaguer

Máster en Ciencias de la Computación

Email: trinchet@uci.cu

Tutor 2:

César Raúl García Jacas

Ingeniero en Ciencias Informáticas

Email: crjacas@uci.cu

Agradecimientos

Agradecer en pocas palabras a todas las personas que han colaborado en la materialización de este trabajo es algo difícil. En primer lugar quiero agradecer a mi familia que ha guiado con sabiduría mi formación, en especial, a mis padres que siempre esperan lo mejor de mí. A mi hermano por motivarme a ser mejor cada día. A mi primo Ramoncito por su apoyo incondicional, a Ricardo por cocinar tan rico. A mi tía Anita por cuidarnos cuando mamá no estaba. A mi abuelas por quererme tanto. A mis compañeros de estudio, los nuevos y los que ya no están. Me disculpan que no mencione sus nombres pero haría muy extensa esta lista. A los profesores que contribuyeron en mi formación, especialmente a Noryis y Andreíta. A Reisel por compartir una gran amistad y por ser mi brazo derecho en los momentos más difíciles. A mis profesoras particulares de Matemática y Física por la ayuda en los primeros años. A mis tutores por la confianza depositada para realizar esta investigación. A Adrián por su invaluable ayuda. A mis familiares aquí en la capital, por cuidarme y brindarme lo mejor en todos estos años. A mi tía Tomasa por recibirme en su casa. Quisiera hacer un reconocimiento especial a Yordán y Yennys por darme un espacio entre ustedes como su hijo en los primeros años, al batallón de Santa Fe por atenderme en el final de la carrera, con mucho cariño a mi tía Dominga, a mi tío Tomás, Tania , Maru, Roge, Papito y Ernesto. A Mabel y Raúl por abrirme las puertas de su casa en Venezuela. A Prieto por confiar siempre en mí. A Delia y Amparito por el apoyo espiritual. A Mayelín por su amistad sincera.

Dedicatoria

*A **MAMI** por ser la estrella que me guía desde los primeros días.*

*A mi **MAMÁ** por su amor y dedicación incondicionales todos estos años.*

*A mi **PAPÁ** por estar siempre a mi lado.*

*A mi **HERMANO** por ser mi fuente de inspiración.*

A mis primos (Ramoncito, Ricardo, Roli, Rafelito y Yordán) que han sido mis hermanos mayores en todo este tiempo, todos ustedes siéntanse ingenieros.

A mis otras madres Yennys y Mabel.

Resumen

El desarrollo científico-técnico alcanzado por la humanidad trae consigo que se presenten problemas de optimización que demandan un alto grado de cómputo para su resolución. La Optimización por Enjambre de Partículas es una técnica relativamente nueva que ha alcanzado buenos resultados en los problemas en los que ha sido aplicada. El presente trabajo se ha enfocado en el estudio de este método y en el desarrollo de una biblioteca eficiente, portable y de propósito general basada en esta técnica que permite resolver problemas de optimización de alto costo computacional de forma secuencial y distribuida sobre la Plataforma de Tareas Distribuidas T-arenal. Se aborda la aplicación de la biblioteca en la resolución de un *K-Means Problem*.

Palabras Clave: Optimización por Enjambre de Partículas, biblioteca, T-arenal, *K-Means Problem*.

Índice

Introducción

El vocablo *optimización* tiene su origen etimológico en óptimo, derivado del latín *optimus* que significa "*el mejor, excelente*", y este, a su vez, del indoeuropeo op-tamo-, superlativo de op- *"producir mucho, trabajar"* (1). La optimización en el sentido de encontrar la mejor solución, o a lo sumo, una solución aceptable para un problema es un campo de gran importancia en la vida real. Un problema de optimización consiste en encontrar la(s) solución(es), de un conjunto de soluciones admisibles, que minimizan o maximizan el rendimiento de un determinado proceso. En cualquier tarea que involucre una toma de decisiones, en el propósito de seleccionar la mejor entre varias disponibles, siempre puede surgir un problema de optimización (2).

Ejemplos hay varios: encontrar una distribución de recursos que permita ejecutar cierta tarea en el menor tiempo posible, determinar el camino más corto entre dos localizaciones, la organización de una agenda, etc. En general, los problemas anteriormente señalados son lo suficientemente pequeños y pueden ser resueltos sin recurrir a elementos externos a nuestro cerebro; sin embargo, en la medida en que se hacen más grandes y complejos[1] el uso de medios de cómputo para tratar de resolverlos resulta necesario.

Las técnicas metaheurísticas representan una familia de técnicas de optimización que han adquirido un auge considerable en las últimas dos décadas debido a su aplicación con buenos resultados en la resolución de problemas complejos. Esta clase de algoritmos incluye técnicas como: Búsqueda Tabú, Enfriamiento Simulado, Algoritmos Géneticos y Optimización por Colonia de Hormigas, etcétera (3; 4; 5; 6).

Aunque el empleo de metaheurísticas permite reducir significativamente la complejidad temporal del proceso de resolución de estos problemas, este tiempo puede seguir siendo muy elevado en algunos problemas reales, esto implica que una única estación de trabajo no satisfaga la demanda de cómputo requerida.

El empleo de computadoras que funcionan como estaciones de trabajo puede ser explotado con el fin de encontrar una solución a un determinado problema "de gran reto" computacional, máxime si se aprovecha el estado ocioso de las mismas, como puede ser en horarios nocturnos o no laborables (7). Con la proliferación de plataformas paralelas de cómputo eficientes la implementación paralela de estas metaheurísticas surge de forma natural como una alternativa para acelerar la obtención de soluciones a estos problemas (8).

[1] Se refiere a que su resolución implica un alto costo computacional.

La *Optimización por Enjambre de Partículas* o *Particle Swarm Optimization*, de aquí en adelante (PSO[2]), es una técnica metaheurística inspirada en el comportamiento social del vuelo de las bandadas de aves o el movimiento de los bancos de peces. PSO fue originalmente formulada por el psicólogo-sociólogo James Kennedy y por el ingeniero Russell Eberhart en 1995 (9). La PSO cuenta con aplicaciones en problemas complejos: Gene Ordering in Microarray Data (GOMAD), Gestión de Área de Localización y Biomecánica, donde se aplican esquemas paralelos-distribuidos de la técnica para resolver estos problemas (10; 11).

La Universidad de las Ciencias Informáticas (UCI), creada en 2002, es sin lugar a dudas el centro a nivel nacional[3] con mayor cantidad de computadoras personales, interconectadas a través de la red universitaria, red que cuenta con una tasa de transferencia de 100Mbps, infraestructura de prestaciones aceptables para soportar este tipo de aplicaciones.
El Centro de Tecnologías de Gestión de Datos (DATEC), adscrito a la Facultad 6 de la UCI, posee en su portafolio de proyectos la Plataforma de Servicios Bioinformáticos. Uno de sus módulos es la Plataforma de Tareas Distribuidas (T-arenal). La Plataforma es un sistema distribuido[4] de propósito general que permite resolver problemas que requieren de altas prestaciones de cómputo, distribuyendo la carga de trabajo entre los diferentes clientes disponibles (12). En estos momentos la Plataforma se encuentra desplegada en la universidad y se ha utilizado para resolver problemas complejos: *docking*[5] molecular (7) y modelado de yacimientos lateríticos (13), obteniéndose buenos resultados. Sin embargo, esta no cuenta con las herramientas necesarias para dar solución eficiente a problemas de optimización, resultando un tanto compleja la modelación y resolución de un problema de este tipo, debido a que es necesario implementar totalmente el problema y el algoritmo que le dará solución, lo que supone un costo en tiempo adicional para encontrar una solución para esta clase de problemas.

En tanto, PSO es una técnica que puede ser distribuida sin mayores inconvenientes debido al paralelismo implícito en su formulación, hecho que la convierte en una técnica competente para ser

[2] En la literatura lo podemos encontrar como población, cúmulo, nube o colmena de partículas. En este trabajo se usará el término “enjambre”.

[3]Se refiere a la República de Cuba.

[4] Conjunto de computadoras autónomas que aparecen integradas ante los usuarios como una sola máquina para resolver determinado problema (44)

[5]Proceso que intenta identificar cuáles moléculas (ligandos) de una base de datos tendrían la capacidad de unirse al sitio activo de la proteína diana (90)

distribuida en la Plataforma T-arenal, lo que permitiría reducir el tiempo en determinar una solución para estos problemas.

En términos de la situación anterior se define el **problema a resolver** como sigue: ¿Cómo resolver problemas de optimización de alto costo computacional según las posibilidades que ofrecen los recursos de cómputo presentes en la Universidad de Ciencias Informáticas?

Para dar respuesta al problema planteado se define como **objeto de estudio** lo siguiente: la técnica Optimización por Enjambre de Partículas en Entornos Distribuidos y el **campo de acción** es la técnica Optimización por Enjambre de Partículas sobre la Plataforma de Tareas Distribuidas T-arenal.

En tal sentido se propone como **objetivo general** el siguiente:

- Desarrollar una biblioteca para resolver problemas de optimización de alto costo computacional aplicando la técnica Optimización por Enjambre de Partículas sobre la Plataforma de Tareas Distribuidas T-arenal.

Para dar cumplimiento al objetivo anteriormente formulado se definen las siguientes **tareas de investigación**:

- Revisión del estado del arte del algoritmo Optimización por Enjambre de Partículas.
- Revisión del estado del arte de los esquemas distribuidos del algoritmo Optimización por Enjambre de Partículas.
- Diseño de la Biblioteca de Optimización por Enjambre de Partículas.
- Implementación de la Biblioteca de Optimización por Enjambre de Partículas.
- Modelado de un *K-Means problem*.
- Resolución de un *K-Means problem*.
- Realización de pruebas de la aplicación de la biblioteca en la Plataforma de Tareas Distribuidas T-arenal.

Hipótesis investigativa:

Si se emplean algoritmos basados en la técnica Optimización por Enjambre de Partículas distribuidos sobre la Plataforma T-arenal para la resolución de problemas de optimización de alto costo computacional, entonces es posible disminuir el tiempo de ejecución necesario para resolverlos.

Variable:

- Tiempo de ejecución.

Métodos científicos:

El método Analítico-Sintético posibilitó realizar un análisis de la literatura viabilizando la extracción de los elementos más relevantes que se relacionan con el objeto de estudio de la presente investigación.

El presente documento está estructurado en 3 capítulos:

Capítulo 1: Fundamento Teórico: En este capítulo se presenta la información obtenida en la revisión bibliográfica realizada sobre el estado del arte de la técnica PSO.

Capítulo 2: Diseño e Implementación de la Biblioteca PSO: En este capítulo se exponen los elementos relativos al diseño e implementación de la biblioteca PSO.

Capítulo 3: Resultados: En este capítulo se presentan y discuten los resultados derivados de la aplicación de la biblioteca a un *K-Means problem*.

Capítulo 1: Fundamento Teórico.

En este capítulo se describen los principales conceptos relativos a la optimización, se abordan las técnicas metaheurísticas y sus particularidades. Además se introduce un tipo concreto de estas técnicas (PSO), se incluye una descripción detallada de esta técnica que comprende su basamento teórico, parámetros del algoritmo, topologías, variantes, esquemas distribuidos, etc.

1.1. Conceptos preliminares de optimización.

Un **problema de optimización sin restricciones** se define como la búsqueda de un punto *x** tal que $f(x^*) \leq f(x)$[6] $\forall x \in S$ donde $f: \mathbb{R}^n \rightarrow \mathbb{R}$ y *S* es el espacio de búsqueda del problema.

Óptimo global: Una solución $s^* \in S$ es un óptimo global si tiene asociado un mejor valor de la función *f* que todas las soluciones del espacio de búsqueda, $\forall s \in S, f(s^*) \leq f(s)$. El principal objetivo de la optimización es encontrar una solución s* que sea un óptimo global (14).

Para la optimización de problemas y cálculos de alta complejidad se han desarrollado múltiples técnicas y métodos. Estas técnicas, en términos de la garantía de optimalidad de la solución que dan para un problema, se pueden taxonomizar[7] en: **exactas** y **aproximadas** (15).
Las **técnicas exactas** (enumerativas, exhaustivas, etcétera) garantizan encontrar la solución óptima para cualquier instancia de cualquier problema[8] en un tiempo acotado. Sin embargo, existen problemas que por su complejidad no pueden ser abordados con éxito por este tipo de técnicas ya que el tiempo necesario para llevarlos a cabo, aunque acotado, crece exponencialmente con el tamaño del problema, llegando a ser en determinados casos de días, meses e incluso años; de ahí se deriva el surgimiento de **técnicas aproximadas** para tratar de solucionarlos, estas deben ser capaces de encontrar una buena solución en un tiempo razonable (aplicaciones en tiempo real).

6 Se asume sin pérdida de generalidad un problema de minimización. Maximizar una función objetivo f es equivalente a minimizar – f.

7 Derivado de taxonomía que constituye la ciencia que trata de los principios, métodos y fines de la clasificación (69).

8 Esta investigación se centra en problemas de decisión (optimización). Existen otro tipo de problemas denominados indecidibles, es decir, no existe un algoritmo que lo resuelva, incluso con recursos ilimitados (88), un ejemplo es el llamado *halting problem* o problema de la parada (83).

1.2. Técnicas aproximadas de optimización.

Dentro de los algoritmos aproximados se distinguen tres tipos: los heurísticos constructivos (ávidos), los métodos de búsqueda local (o métodos de seguimiento del gradiente) y las metaheurísticas.

Los *heurísticos constructivos* suelen ser los métodos más rápidos. Generan una solución partiendo de una vacía a la que se le va añadiendo componentes hasta tener una solución completa, que es el resultado del algoritmo. Aunque en la mayoría de los casos encontrar un heurístico constructivo es relativamente fácil, las soluciones ofrecidas suelen ser de muy baja calidad, y encontrar métodos de esta clase que produzcan buenas soluciones es muy difícil ya que dependen mucho del problema, y para su planteamiento se debe tener un conocimiento muy extenso del mismo. Además, en muchos problemas es casi imposible, por ejemplo, en aquellos con muchas restricciones puede que la mayoría de las soluciones parciales sólo conduzcan a soluciones no factibles (8; 16).

Los *métodos de búsqueda local* parten de una solución ya completa junto con el uso del concepto de vecindario, recorren parte del espacio de búsqueda hasta encontrar un óptimo local[9]. El vecindario de una solución *s*, que se denota como *N*(*s*), es el conjunto de soluciones que se pueden construir a partir de *s* aplicando un operador específico de modificación. Estos métodos, partiendo de una solución inicial, examinan su vecindario y eligen el mejor vecino continuando el proceso hasta que encuentran un óptimo local. En muchos casos, la exploración completa del vecindario es inabordable y se siguen diversas estrategias, dando lugar a diferentes variaciones del esquema genérico. Según el operador de movimiento elegido, el vecindario cambia y el modo de explorar el espacio de búsqueda también, pudiendo simplificarse o complicarse el proceso de búsqueda (8).

Finalmente, en los años setenta surgió una nueva clase de algoritmos no exactos, cuya idea básica era combinar diferentes métodos heurísticos a un nivel más alto para conseguir una exploración del espacio de búsqueda de forma eficiente y efectiva. Estas técnicas se han denominado *metaheurísticas*.

1.3. Técnicas metaheurísticas.

El término heurística proviene del griego *heuriskein*, que significa “hallar, inventar”, a su vez, el prefijo *meta* también de origen griego cuyo significado es “más allá” o “a un nivel superior”. El término

[9] Un óptimo local es una solución mejor o igual que cualquier otra solución de su vecindario.

metaheurística fue introducido por F. Glover. Las metaheurísticas se pueden definir como metodologías generales de alto nivel que sirven de guía para el diseño de heurísticas subyacentes para resolver problemas de optimización (17).

De las diferentes descripciones de metaheurísticas que se encuentran en la literatura se pueden destacar ciertas propiedades fundamentales que caracterizan a este tipo de métodos (8; 18; 19):

- Las metaheurísticas son estrategias que guían el proceso de búsqueda.
- El objetivo es una exploración del espacio de búsqueda eficiente para encontrar soluciones (casi) óptimas.
- Las metaheurísticas son algoritmos no exactos y generalmente son no deterministas.
- Pueden incorporar mecanismos para evitar las áreas del espacio de búsqueda no óptimas.
- El esquema básico de cualquier metaheurística es general y no depende del problema a resolver.
- Las metaheurísticas hacen uso de conocimiento del problema que se trata resolver en forma de heurísticos específicos que son controlados de manera estructurada por una estrategia de más alto nivel.
- Las metaheurísticas utilizan funciones de bondad[10] (funciones de *fitness*) para cuantificar el grado de adecuación de una determinada solución.

En resumen, una metaheurística es una estrategia de alto nivel que aplica diferentes métodos para explorar el espacio de búsqueda. Resulta de especial interés que en el proceso de exploración exista equilibrio entre *diversificación* e *intensificación*. El término diversificación se refiere a la exploración del espacio de búsqueda, en tanto, intensificación se refiere a la explotación de algún área específica de ese espacio. El equilibrio entre estos dos aspectos contrapuestos es de gran importancia, ya que por un lado deben identificarse rápidamente la regiones prometedoras del espacio de búsqueda global y por el otro lado no se debe malgastar tiempo en las regiones que ya han sido exploradas o que no contienen soluciones de alta calidad.

Existen varios criterios de clasificación para las metaheurísticas. De acuerdo a las características que se seleccionen se pueden obtener diferentes taxonomías: basadas o no en la naturaleza, con o sin memoria, deterministas o estocásticas, basadas en solución única (trayectoria, S-metaheurísticas) o en población (P-metaheurísticas), etc (15). Para los propósitos de este trabajo se elige la última taxonomía la cual es ampliamente utilizada por la comunidad científica.

[10]En ocasiones se denomina costo, utilidad, aptitud o fitness.

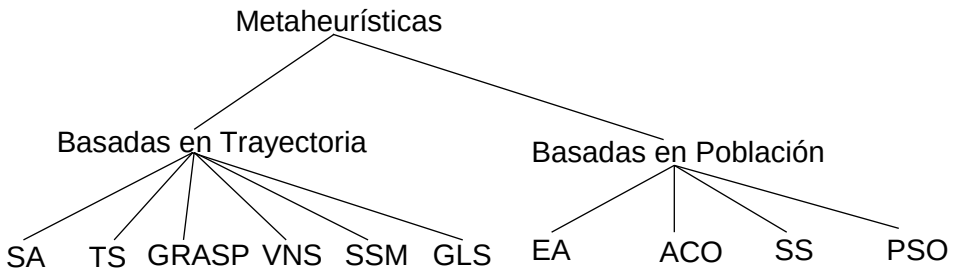


Figura 1.1 Clasificación de las Metaheurísticas.

1.3.1. Metaheurísticas Basadas en Trayectoria.

La principal característica de estos métodos es que parten de un punto y mediante la exploración del vecindario van actualizando la solución actual, formando una trayectoria. La mayoría de estos algoritmos surgen como extensiones de los métodos de búsqueda local simples a los que se les añade alguna característica para escapar de los óptimos locales. Normalmente se termina la búsqueda cuando se alcanza un número máximo predefinido de iteraciones, se encuentra una solución con una calidad aceptable, o se detecta un estancamiento del proceso. Vale resaltar que estos métodos están orientados a la explotación del espacio de búsqueda (15).

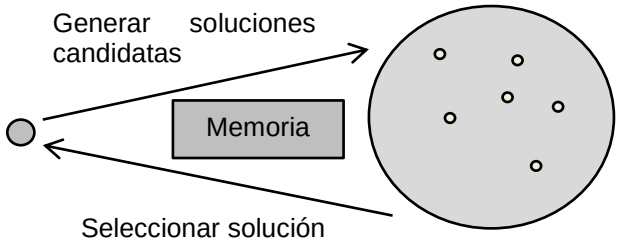


Figura 1.2 Principios fundamentales de las Metaheurísticas basadas en Trayectoria.

Dentro de estas técnicas se encuentran las siguientes:

- *Enfriamiento Simulado o Simulated Annealing (SA)* (4): es una de las más antiguas metaheurísticas y posiblemente es el primer algoritmo con una estrategia explícita para escapar de los óptimos locales. La idea del SA es simular el proceso de recocido del metal y del cristal. Para evitar quedar atrapado en un óptimo local, el algoritmo permite elegir una solución peor que la solución actual.

- *Búsqueda Tabú o Tabu Search* (TS) (3): es una de las metaheurísticas que se ha aplicado con más éxito en la resolución de problemas de optimización combinatoria[11]. Un elemento esencial en este método es el uso explícito de un historial de la búsqueda (una memoria de corto plazo), tanto para escapar de los óptimos locales como para implementar su estrategia de exploración y evitar buscar varias veces en la misma región.
- *Procedimiento de Búsqueda Miope Aleatorizado y Adaptativo o Greedy Randomized Adaptive Search Procedure* (GRASP) (20): es una metaheurística para la resolución de problemas combinatorios que combina heurísticos constructivos con búsqueda local. GRASP es un procedimiento iterativo compuesto de dos fases: primero una construcción de una solución y después un proceso de mejora.
- *Búsqueda con Vecindario Variable o Variable Neighborhood Search* (VNS) (21): es un método propuesto por P. Hansen y N. Mladenovic, la idea fundamental de VNS es la exploración de un conjunto de vecindarios predefinidos para obtener una mejor solución. VNS explota el hecho de que usando varios vecindarios para la búsqueda local se generan diferentes óptimos locales y que el óptimo global es el óptimo local para un determinado vecindario.
- *Búsqueda por Métodos de Alisado o Search Smoothing Methods* (SSM) (22): consiste en la modificación del espacio de búsqueda asociado a un problema determinado con el objetivo de reducir la cantidad de óptimos locales. La idea principal de este método es dividir el problema en instancias de complejidad inferior que el original las cuales al ser resueltas proveerán soluciones óptimas locales que guiarán el proceso de búsqueda.
- *Búsqueda Local Guiada o Guided Local Search* (GLS) (23): es una metaheurística que ha sido aplicada a problemas de optimización combinatorios. El principio básico de GLS es la actualización dinámica de la función objetivo de acuerdo al óptimo local generado anteriormente.

1.3.2. Metaheurísticas Basadas en Población.

Los métodos basados en población se caracterizan por trabajar con una población inicial de soluciones. En cada iteración se realiza un proceso de generación de una nueva población así como el reemplazo de la generación actual (15).

[11] Las variables asociadas a las soluciones de este tipo de problema son de dominio discreto.

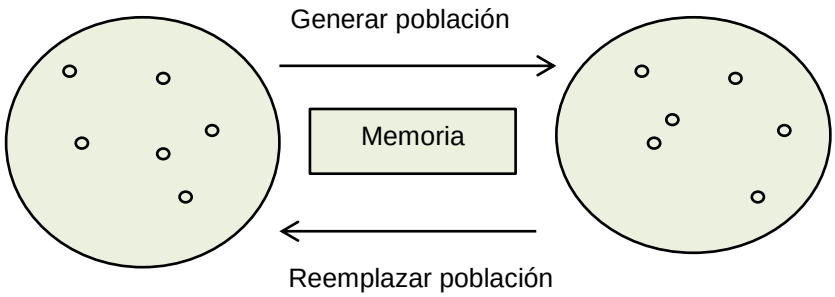


Figura 1.3 Principios fundamentales de las Metaheurísticas poblacionales.

En la etapa de generación se crea una nueva población así como en la fase de reemplazo se realiza una selección entre los individuos que forman parte de la población anterior y la generada actualmente para conformar la nueva población. Este proceso se lleva a cabo hasta que se cumpla un criterio de parada definido. Los procesos de generación y reemplazo de la población pueden ser *sin memoria*, es decir, solo se tiene en cuenta la población actual para realizar los mismos. Sin embargo, es posible tener en cuenta el desempeño histórico de la población para la realización de estos procesos.
La mayoría de los métodos basados en población son algoritmos basados en procesos naturales. La diferencia fundamental entre estas técnicas radica en la forma en que se realizan los procesos de *generación*, *selección* y la *memoria de búsqueda* que se utiliza durante este proceso.

El siguiente pseudocódigo describe de forma general el funcionamiento de este tipo de metaheurísticas.

Procedimiento General de las Metaheurísticas Basadas en Población

Generación de la población inicial.
$P \leftarrow P_0$
$k \leftarrow 0$
Repetir
Generación de una nueva población
Generar (P_k')
Seleccionar nueva población
$P_{k+1} \leftarrow$ ***Seleccionar_Individuos (P_k U P_k')***
$k \leftarrow k+1$

Hasta Criterio de Parada
Salida: Mejor solución (es) encontrada(s)

- **Generación**: se realiza la generación de una nueva población, de acuerdo con la estrategia de generación las metaheurísticas puede ser clasificadas en dos grupos:

1. Basadas en la evolución: En esta taxonomía las soluciones que componen la población se seleccionan, se reproducen usando operadores de modificación que actúan directamente sobre su representación. La nueva solución se construye por medio de los diferentes atributos de las soluciones que forman la población actual.
2. Basadas en Blackboard[12]: En esta categoría las soluciones participan en la construcción de memoria compartida, la cual será la principal entrada al proceso de generación de las nuevas soluciones. En este caso la recombinación entre las soluciones se realiza de forma indirecta a través de la memoria compartida.

- **Selección**: se realiza la selección de la nueva población la cual estará compuesta por la unión de la población actual y la población generada. Una estrategia tradicional de selección consiste en escoger la población generada como la nueva población, otras estrategias emplean algún tipo de elitismo para obtener las mejores soluciones de los dos conjuntos. En el caso de las basadas en Blackboard no existe un criterio explícito de selección.
- **Memoria de búsqueda**: representa la información que se extrae y se *memoriza* en la búsqueda, esta información varía de una técnica a otra. En general la memoria de búsqueda se limita solo a la población de soluciones, sin embargo, existen técnicas donde se memoriza otra información.

Ejemplos de estas técnicas son las siguientes:

- *Algoritmos Evolutivos o Evolutionary Algorithms* (5; 24) están inspirados en la capacidad de la naturaleza para evolucionar seres para adaptarlos a los cambios de su entorno. Esta familia de técnicas siguen un proceso iterativo y estocástico que opera sobre una población de individuos. Cada individuo representa una solución potencial al problema que se está resolviendo, además se le asocia, por medio de una función de aptitud (*fitness*), una medida de su bondad con respecto al problema bajo consideración. Este valor es la información cuantitativa que el algoritmo usa para guiar su búsqueda. En los métodos que siguen el esquema de los algoritmos evolutivos, la

[12] Un blackboard es una base de conocimiento compartida, la cual es actualizada iterativamente por un grupo de agentes (70).

evolución de la población se lleva a cabo mediante tres operadores: selección, cruzamiento y mutación.

- *Búsqueda Dispersa o Scatter Search (SS)* (25) se basa en mantener un conjunto relativamente pequeño de soluciones tentativas (llamado conjunto de referencia) que se caracteriza tanto por contener buenas soluciones como soluciones diversas. Este conjunto se divide en subconjuntos de soluciones a las cuales se les aplica una operación de recombinación y mejora. Para realizar la mejora o refinamiento de soluciones se suelen utilizar mecanismos de búsqueda local.
- *Optimización por Colonias de Hormigas o Ant Colony Optimization (ACO)* (6) está inspirada en el comportamiento de las hormigas naturales cuando realizan la búsqueda de comida. Este comportamiento es el siguiente: inicialmente, las hormigas exploran el área cercana a su nido de forma aleatoria. Tan pronto como una hormiga encuentra la comida, la lleva al nido. Mientras que realiza este camino, la hormiga va depositando una sustancia química denominada feromona. Esta sustancia ayudará al resto de las hormigas a encontrar la comida. Esta comunicación indirecta entre las hormigas mediante el rastro de feromona las capacita para encontrar el camino más corto entre el nido y la comida. En esta técnica, el rastro de feromona es simulado mediante un modelo probabilístico. Este comportamiento es el que intenta simular este método para resolver problemas de optimización.
- *Optimización por Enjambre de Partículas o Particle Swarm Optimization* (9) son técnicas metaheurísticas estocásticas basadas en poblaciones e inspiradas en el comportamiento social del vuelo de bandadas de aves o el movimiento de los bancos de peces. La simulación de este comportamiento da lugar a un método para resolver problemas de optimización.

  Esta técnica fue seleccionada para ser implementada en la presente investigación debido a dos aspectos fundamentales: el primero está relacionado con el teorema *Non Free Lunch* (26) para algoritmos de optimización y búsqueda que plantea que ningún algoritmo por sofisticado que sea tiene un desempeño superior a ningún otro en todos los problemas posibles.

  El segundo elemento tiene que ver con la efectividad del método en la resolución de problemas de optimización de naturaleza continua, esto permitiría al proyecto productivo enfrentar mayor variedad de problemas complejos de optimización debido a que están en desarrollo otras soluciones basadas en Algoritmos Genéticos y Colonia de Hormigas respectivamente. Estas propuestas pudieran incluso hibridarse para obtener soluciones de mayor calidad y precisión.

1.4. Optimización por Enjambre de Partículas.

La Optimización por Enjambre de Partículas, nace, al igual que otras técnicas estocásticas del cálculo evolutivo, en un intento por imitar el comportamiento de procesos naturales. Los orígenes de PSO como método estocástico de optimización global se remontan a los estudios iniciados por Kennedy y Eberhart, quienes se fijan como objetivo inicial simular gráficamente el movimiento sincronizado e impredecible de grupos tales como los bancos de peces o las bandadas de aves, intrigados por la capacidad de estos grupos para separarse, reagruparse o encontrar alimento (9).

En línea con trabajos previos en el ámbito de la biología y de la sociología, que concluyen que el comportamiento, inteligencia y movimiento de estas agrupaciones, entre las cuales se podría incluir con un cierto grado de abstracción a los seres humanos, está relacionado directamente con la capacidad de los individuos para compartir información y aprovecharse de la experiencia acumulada por sus congéneres (27; 28),en (9) se modela dicho comportamiento de forma matemática utilizando expresiones simples que revelan su potencial como método de optimización.

En la terminología utilizada en PSO, Kennedy y Eberhart introducen el término general partícula o agente para representar a los peces, pájaros, abejas, hormigas o cualquier otro tipo de individuos que exhiban un comportamiento social como grupo, en forma de una colección de agentes que interactúan entre sí. De acuerdo con los fundamentos teóricos del método, el movimiento de cada una de estas partículas hacia un objetivo común está condicionado por dos factores básicos, la memoria autobiográfica de la partícula o nostalgia y la influencia social de todo el enjambre (9; 29).

El enjambre de partículas (*swarm*) es un sistema multiagente, es decir, las partículas son agentes simples que se mueven por el espacio de búsqueda, guardan (y posiblemente comunican) la mejor solución que han encontrado (10; 30).

Figura 1.4 Enjambres en la naturaleza.

1.4.1. Descripción del Algoritmo de Optimización por Enjambre de Partículas.

En el modelo básico de PSO el enjambre está compuesto por un conjunto de partículas que *sobrevuelan* el espacio D-dimensional.

En general, una partícula está compuesta por:

- Vector $p_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,d})$: representa la posición (solución) actual de la partícula.
- Vector $v_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,d})$: representa la velocidad actual de la partícula.
- Vector $pbest_i = (pbest_{i,1}, pbest_{i,2}, \ldots, pbest_{i,d})$: almacena la mejor posición (solución) encontrada por la partícula hasta ese momento.
- Valor de aptitud $bestFitness_i$: almacena el valor asociado a la mejor solución encontrada por la partícula *i* hasta ese instante.
- Valor de aptitud $fitness_i$: almacena el valor asociado a la solución actual encontrada por la partícula.

donde:

- i: *i-ésima* partícula del enjambre.
- d: dimensiones del espacio de búsqueda.

El enjambre se inicializa generando las posiciones p_i de las partículas de forma aleatoria, regular o heurística, posteriormente se actualizan los valores $fitness_i$ y $bestFitness_i$ respectivamente. Según Clerc (31) es posible obtener mejoras significativas en la diversidad inicial del enjambre de partículas empleando diferentes métodos para construir el enjambre inicial, por ejemplo empleando la distribución Hammersley (32).

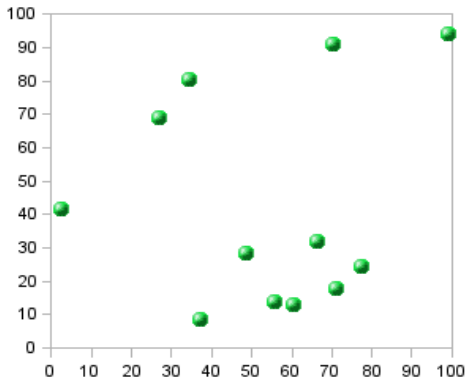


Figura 1.5 Inicialización aleatoria de acuerdo a una distribución uniforme.

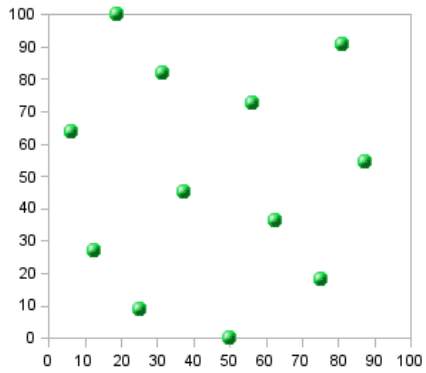


Figura 1.6 Inicialización de acuerdo a una distribución Hammersley.

Las (figuras 1.5 y 1.6) muestran dos tipos de distribuciones para inicializar las posiciones de las partículas que conformarán el enjambre, nótese una mejora significativa en cuanto al muestreo del espacio de búsqueda en la distribución de la (figura 1.6).

En tanto las velocidades v_i se generan en el intervalo $[-v_{máx}, v_{máx}]$, donde $v_{máx}$ representa la velocidad máxima que podrá alcanzar una partícula **(31)**.

1.4.2. Operador velocidad y parámetros del algoritmo.

En la formulación de PSO se define la velocidad de partícula como el único operador disponible para controlar la evolución de la optimización (9). El vector velocidad se actualiza en cada iteración teniendo en cuenta 3 componentes: la velocidad anterior (inercia), un componente *cognitivo*[13] y un componente *social*. El modelo matemático resultante y que constituye el núcleo conceptual del algoritmo PSO está representado por las siguientes ecuaciones (10):

$$v_i(k+1) = wv_i(k) + c_1 rand1\big(pBest_i - p_i\,(k)\big) + c_2 rand2\big(pgBest_i - p_i\,(k)\big)\ (1)$$

$$p_i(k+1) = p_i(k) + v_i(k+1)\ (2)$$

donde:

- $v_i\,(k)$: velocidad de la partícula *i* en la iteración *k.*
- *w: factor inercia.*
- c_1, c_2: coeficientes de aprendizaje que controlan la influencia de los factores cognitivo y social respectivamente.
- $p_i\,(k)$: posición de la partícula *i* en la iteración *k.*

[13] También conocido como experiencia personal o memoria autobiográfica.

- *rand1, rand2*: números aleatorios uniformemente distribuidos en el intervalo [0,1] para emular el comportamiento estocástico y un tanto impredecible que exhibe la población del enjambre.
- $pBest_i$: mejor posición (solución) encontrada por la partícula *i* hasta el momento.
- $pgBest_i$: mejor posición (solución) del entorno de la partícula *i,* (*lBest o localBest*) o de todo el enjambre (*gBest o globalBest*) encontrada hasta el momento.

La ecuación (1) representa la actualización del vector velocidad de la partícula i en la iteración k, el componente cognitivo lo representa el factor $c_1 rand1\left(pBest_i - p_i\,(k)\right)$ cuyo significado es la distancia entre la mejor posición encontrada por la partícula y su posición actual, es decir, la decisión que tomará la partícula según la influencia de su propia experiencia.
El componente social lo representa $c_2 rand2\left(pgBest_i - p_i\,(k)\right)$ que análogamente representa la distancia entre la posición de la mejor partícula de su vecindario y la posición actual de la partícula, este componente representa la decisión que tomará partícula de acuerdo con la influencia que el entorno social ejerce sobre ella.

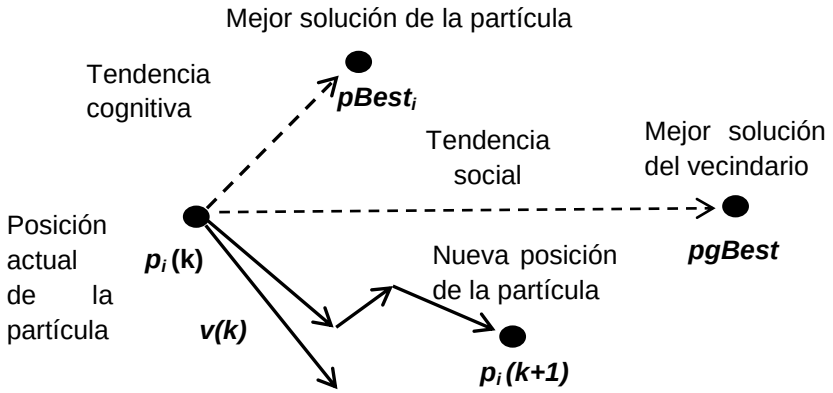


Figura 1.7 Movimiento de la partícula en el espacio de soluciones.

El factor inercia *w* controla el impacto de la velocidad anterior con la respecto a la actual, representa un equilibrio entre la capacidades de exploración y explotación del enjambre; valores elevados de *w* favorecerán la exploración del espacio de soluciones, mientras que valores más pequeños de *w* incrementarán la explotación de una región determinada del espacio.

Con el objetivo de lograr este balance Shi y Eberhart proponen iniciar con *w= 0.9* e ir disminuyendo linealmente hasta *w=0.4* (33).
El factor de inercia puede ser calculado como:

$$w = w_{max} - \frac{w_{max} - w_{min}}{iter_{max}} iter \quad (3)$$

donde:

- w_{max} : peso inercial inicial.
- w_{min} : peso inercial final.
- $iter_{max}$: cantidad máxima de iteraciones del algoritmo.
- *iter*: iteración actual del algoritmo.

Existen en la literatura varias formas de calcular la velocidad de la partícula, una alternativa al modelo anteriormente propuesto es el denominado *factor de constricción* introducido por Clerc y Kennedy, el cual está regido por las ecuaciones:

$$v_i(k+1) = \chi[v_i(k) + c_1 rand1\big(pBest_i - p_i\,(k)\big) + c_2 rand2\big(pgBest_i - p_i\,(k)\big)] \quad (4)$$

$\chi = \frac{2\delta}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|}$, donde: $\delta \in [0,1]$, $\varphi = c_1 + c_2, \varphi > 4$ (5)

En este modelo puede obviarse el parámetro $v_{máx}$, aunque en la práctica se suele combinar esta estrategia con dicho parámetro (34).

Otra alternativa propuesta por Mendes y Kennedy (35) donde los *k* vecinos de cada partícula contribuyen en la actualización de su velocidad, es la denominada *Enjambre de Partículas Totalmente Informado* o *Fully Informed Particle Swarm* (FIPS).
La ecuación de velocidad quedaría como sigue:

$$v_i(k+1) = \chi[v_i(k) + \frac{1}{K_i}\sum_{n=1}^{K_i} U(0,\varphi)\,\big(pkBest_i - p_i\,(k)\big)]$$

donde:

χ: factor de constricción.

φ: suma de los factores cognitivo y social.

$pkBest_i$: la mejor posición del vecino i.

p_i: posición actual de la partícula.

El siguiente pseudocódigo describe el algoritmo clásico de PSO:

Algoritmo Clásico PSO

k←0

I←Inicializar Enjambre de partículas;

Repetir

k←k+1

Para cada partícula i=1,…, I

Evaluar $f(p_i)$

Si $fitness_i$ es mejor $bestFitness_i$ ***entonces***

$pBest_i \leftarrow p_i$

$bestFitness_i \leftarrow fitness_i$

Fin Si

Si $fitness_i$ es mejor $gBestFitness_i$ ***entonces***

$pgBest \leftarrow p_i$

$gBestFitness \leftarrow fitness_i$

Fin Si

Fin Para

Para todas las partículas i

Actualizar velocidad

Actualizar posición

Fin Para

Hasta Criterio de Parada

Salida: La mejor solución encontrada.

1.4.3. Topologías del Enjambre de Partículas.

En PSO las partículas mejoran sus aptitudes imitando los comportamientos y tendencias de los mejores congéneres de la población. El establecimiento del entorno de una partícula, o lo que es lo mismo, qué otras partículas le influencian, tiene una trascendencia vital en el rendimiento del algoritmo. Las topologías definen el entorno de interacción de una partícula con su vecindario[14], cada partícula se incluye en su propio vecindario. En dependencia de la topología que adquiera la población, la

[14] No debe confundirse con el concepto de vecindario de una solución utilizado previamente en este capítulo. En este contexto la vecindad de una partícula es un conjunto de partículas.

transmisión de la información entre individuos puede acelerarse o ralentizarse, lo cual está estrechamente relacionado con la velocidad de convergencia y con la capacidad del algoritmo para escapar de soluciones locales (10).

En general, los entornos o topologías pueden ser de dos tipos:

- Geográficos: se escogen las partículas más cercanas (generalmente se toma la distancia euclídea como medida de proximidad) para formar la vecindad.
- Sociales: se define a priori una lista de vecinos de la partícula, con independencia de su posición en el espacio.

Las sociometrías son las más utilizadas y a su vez pueden ser clasificadas en 2 modelos fundamentales: *GBest y LBest*.

En el modelo GBest se define todo el enjambre como la vecindad de cada partícula, lo que garantiza que cada partícula sea influenciada por la mejor solución encontrada en todo el enjambre favoreciendo la explotación del espacio de soluciones, esto provoca una rápida convergencia a la solución. Este modelo tiene el inconveniente de ser sensible a quedar atrapado en óptimos locales.
En el caso de LBest cada partícula *orienta* su movimiento bajo la influencia de *k* vecinos, quedando aislada, en cierta medida, de las partículas más alejadas del enjambre, lo que implica que la convergencia hacia la solución sea más lenta, siendo a su vez menos propenso a *caer* en óptimos locales, lo que no significa que sea inmune a este fenómeno (36).
Estudios recientes proponen una configuración intermedia llamada topología de Von Neumann o Cuadrada (figura 1.8-d). En este caso se dispone el enjambre como una matriz rectangular, por ejemplo de 5 *x* 4 para un enjambre de tamaño 20, donde cada partícula es conectada con las partículas superior, inferior, izquierda y derecha (37).
A pesar de la variedad de sociometrías puede elegirse el modelo geográfico para establecer la vecindad de la partícula, sin embargo, este tiene como inconveniente un costo computacional adicional (38). Se puede afirmar que no existe una topología óptima para cualquier problema, las que para algunos casos han dado buenos resultados en otros han resultado ineficientes.

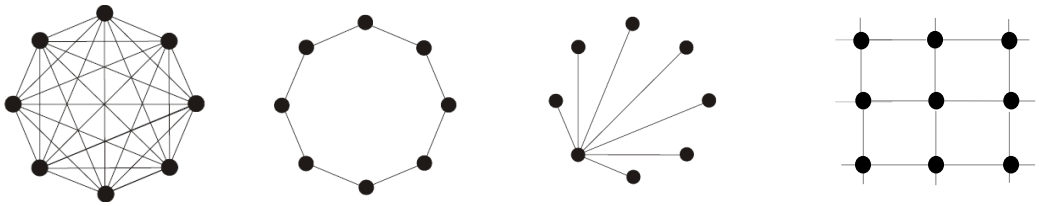


Figura 1.8 Topologías de PSO. (a) Global Best, (b) Local Best (Anillo), (c) Local Best (Abanico), (d) Von Newman

1.4.4. Principales variantes de PSO.

PSO fue concebido originalmente para problemas de optimización continua (9), sin embargo, el método ha sido objeto desde su surgimiento de una constante atención por parte de la comunidad científica internacional lo que ha posibilitado la creación de diferentes variantes o adaptaciones del método para resolver otro tipo de problemas (39).

PSO para problemas continuos.

En esta versión de PSO las partículas representan puntos en el espacio D-dimensional, la evolución de la población es vista como el movimiento de las partículas por el espacio.

Representación de las Partículas

Cada partícula *i* en cualquier versión del PSO está formada como mínimo por tres componentes: *i*:= (*p, pBest, v*).

donde:

- *p:* es un vector real que representa un punto (posición) en el espacio de búsqueda, en la mayoría de los casos, una solución.
- *pBest:* representa la mejor posición (solución) encontrada por la partícula *i* hasta el momento.
- *v:* representa la velocidad de la partícula.

Para ver cómo se realiza la actualización de la velocidad y posición de cada partícula en esta variante respectivamente remitirse a (epígrafe 1.4.2).

PSO para problemas discretos.

En este modelo la partícula solo tiene que decidir sí o no, verdadero o falso, incluye o excluye, etcétera.

Esta versión de PSO difiere del modelo continuo en dos elementos fundamentales:

- **Codificación de las posiciones de las partículas**: generalmente se emplean codificaciones binarias o permutaciones.
- **Modelos de velocidad**: pueden ser de valores reales, estocásticos, etc.

En el modelo estocástico la velocidad se define en términos de la probabilidad de que un bit tome valor 1. En este modelo la velocidad debe estar en el rango [0,1] para lograrlo se utiliza la función:

$$v(k+1)_i = sig\left(v_{i,d}\right)\frac{1}{1+\exp\left(-v_{i,d}\right)} (5)$$

La nueva posición de la partícula se obtiene mediante la ecuación:

$$x_i(k+1) = \begin{cases} 1 \; si \; r_{ij} < sig\left(v_{ij}\,(k+1)\right) \\ 0 \quad en\ otro\ caso \end{cases} (6)$$

donde: r_{ij} es un número aleatorio en el rango [0,1] (40).

Otras variantes de PSO.

Existen diferentes variantes de PSO según (39) hasta 2008 se reconocían cerca de 95 adaptaciones basadas en esta técnica. Entre ellas se encuentran:

- *Adaptive PSO (APSO)*: se basa en la idea de sustituir las partículas que se encuentran *inactivas,* es decir*,* aquellas cuya posición espacial no cambia mucho, por otras nuevas, de forma tal que se mantengan las relaciones que existían antes del reemplazo.
- *Attractive Repulsive Particle Swarm Optimization (ARPSO)*: su objetivo fundamental es erradicar la rápida convergencia de PSO, incluye dos fases: repulsión y atracción.
- *Genetic PSO (GPSO)*: este método se deriva del PSO original y además incluye los mecanismos de reproducción genéticos: cruzamiento y mutación.
- *Species in a Particle Swarm Optimizer (SPSO):* se basa en la división del enjambre en varios subenjambres. El criterio de agrupación, puede ser, por ejemplo: la distancia euclídea. Cada uno de estos subenjambres se denominan *especies*, cada especie tendrá una partícula líder (la mejor adaptada) a la cual se le asociará un radio. Las partículas se *mueven* en el radio de su especie lo

que provoca múltiples evaluaciones en paralelo. Este método se aplica a problemas de optimización multiobjetivo.

1.5. Introducción a la Computación Paralela y Distribuida.

En este apartado se expone una breve reseña sobre la Computación Paralela y Distribuida. Se abordan las principales arquitecturas paralelas así como las principales características de los sistemas distribuidos enfatizando en su capacidad para resolver problemas complejos. En último lugar se presentan algunos índices empleados para medir el rendimiento de las aplicaciones en este tipo de entornos.

1.5.1. Computación Paralela.

La Computación Paralela es una rama de la Ciencia de la Computación que estudia cómo resolver un problema haciendo uso de un conjunto de procesadores que son capaces de trabajar de forma cooperativa (41). Se basa en el principio de que el proceso de solucionar un problema usualmente puede dividirse en pequeñas tareas que pueden ejecutarse simultáneamente y de forma coordinada. De aquí, se deriva el hecho de que se utilice para lograr disminuir el tiempo de ejecución en la solución de problemas costosos computacionalmente y permitir el aumento de la dimensión del problema que puede resolverse.

Arquitecturas paralelas.

Michael Flynn, profesor de ingeniería eléctrica de la Universidad de Stanford, propuso una clasificación para las arquitecturas paralelas basándose en dos criterios: el número de flujos de instrucciones y el número de flujos de datos, obteniendo 4 categorías (42):

- SISD (*Single Instruction Stream-Single Data Stream*): Modelo secuencial tradicional. Ordenador con arquitectura monoprocesador que no explota el paralelismo en las instrucciones ni en los datos. En la actualidad esta clase tiende a desaparecer, dado porque los procesadores que componen las estaciones de trabajo actuales son multinúcleos.
- SIMD (*Single Instruction Stream-Multiple Data Stream*): a diferencia de SISD, en este caso se tienen múltiples procesadores que sincronizadamente ejecutan la misma secuencia de instrucciones, pero en diferentes datos. El tipo de memoria que estos sistemas utilizan es distribuida. Aquí hay N secuencias de datos, una por procesador, así que diferentes datos pueden ser utilizados en cada procesador. Los procesadores operan sincronizadamente y un reloj global se

utiliza para asegurar esta operación. Es decir, en cada paso todos los procesadores ejecutan la misma instrucción, cada uno con diferente dato.

- MISD (*Multiple Instruction Stream-Simple Data Stream*): en este modelo, secuencias de instrucciones pasan a través de múltiples procesadores. Diferentes operaciones son realizadas en diversos procesadores. N procesadores, cada uno con su propia unidad de control comparten una memoria común y una secuencia de datos.
- MIMD (*Multiple Instruction Stream-Multiple Data Stream*): consiste en el modelo más general y puede ser programado para funcionar como uno de los tres anteriores. Este tipo de computadora es paralela y no tiene un reloj central. Cada procesador puede ejecutar su propia secuencia de instrucciones y tener sus propios datos. Los sistemas de cómputo distribuido clasifican dentro de este grupo.

1.5.2. Computación Distribuida y Sistemas Distribuidos.

La Computación Distribuida brinda acceso transparente al poder de cómputo y almacenamiento de muchas computadoras independientes que un usuario necesita para realizar una determinada tarea, al tiempo que permite alcanzar altos índices de rendimiento y confiabilidad mediante el aprovechamiento de esos recursos computacionales (43; 44). El interés por la computación distribuida ha crecido rápidamente en la última década con objetivos diversos según la necesidad. Desde el punto de vista de los usuarios, estos sistemas han sido ampliamente usados para afrontar problemas que requieren computación intensiva en diversas áreas de la ciencia. Los Sistemas Distribuidos representan actualmente una alternativa importante para enfrentar problemas que demandan gran cantidad de cómputo. Ejemplos hay varios: búsqueda de inteligencia extraterrestre (45), simulaciones complejas del clima (46), simulaciones a partir del genoma humano (47), etc. Los componentes del sistema no son más que hardware y software que se comunican entre sí a través de una red, preferiblemente canales de alta velocidad (48; 44).

1.5.3. Evaluación del rendimiento.

En esta sección se presentarán algunos índices de prestaciones utilizados en la computación paralela y distribuida.

- **Ganancia de Velocidad** (Speed-up): El *Speed-up* para *p* procesadores (*Sp)* es el cociente entre el tiempo de ejecución de un programa secuencial (*Ts)* y el tiempo de ejecución de la versión paralela de dicho programa en *p* procesadores (*Tp).*Indica la ganancia de velocidad que se ha obtenido con la ejecución en paralelo.

$$Sp = \frac{Ts}{Tp}$$

- **Eficiencia:** La eficiencia es el cociente entre el speed-up y el número de procesadores. Mide el grado de aprovechamiento de los procesadores para la resolución del problema.

$$E = \frac{Sp}{p}$$

1.6. Optimización por Enjambre de Partículas en Entornos Distribuidos.

La distribución de la técnica Optimización por Enjambre de Partículas para resolver problemas complejos de optimización tiene un gran impacto en problemas que demandan intenso cómputo debido a que permite reducir el tiempo necesario para resolverlos. A continuación se exponen las principales alternativas para realizar la distribución de PSO.

1.6.1. Modelos con población global.

En estos modelos el esquema básico del algoritmo se mantiene igual que el secuencial, por tanto, tendremos una sola población global. El objetivo reside en repartir la carga de trabajo entre varios procesadores.

- **PSO Global (Maestro-Esclavo).**

El proceso maestro reparte las tareas (subgrupos del enjambre original) a los procesos esclavos disponibles, los cuales se encargan de ejecutar la función de evaluación, actualizar las velocidades y posiciones de las partículas así como devolverlas al proceso maestro. El proceso maestro es el encargado de seleccionar la(s) mejor(es) soluciones encontradas. En este modelo la ganancia computacional se ve afectada por el costo de las comunicaciones, por lo que este enfoque resulta particularmente útil cuando el costo de las tareas enviadas a los esclavos es relativamente alto (49).

- **Variante asíncrona: Optimización por Enjambre de Partículas Paralela Asíncrona.**

En este esquema el proceso maestro después de construir el enjambre inicial, emplea una estrategia FIFO para distribuir las partículas de esta forma en cada instante las partículas se actualizarán con la información disponible. Este comportamiento implica que no se obtengan los mismos resultados con respecto al algoritmo secuencial. El orden en que las partículas serán evaluadas cambiará en dependencia de la velocidad con que cada procesador complete la función de evaluación. A pesar de

la heterogeneidad de las tareas y/o recursos computacionales esta estrategia garantiza que cada partícula realice aproximadamente igual cantidad de evaluaciones de la función objetivo. Por su parte los procesos esclavos evalúan la función objetivo y devuelven el resultado al maestro (11).

1.6.2. Modelo de Islas.

En este caso cada isla supone una instancia de nuestro algoritmo, que ejecuta de forma secuencial el mismo problema. Cada población local es inicializada independientemente, de manera que cada isla partirá de poblaciones distintas y obtendrá resultados diferentes. Al final de la ejecución, las poblaciones de cada una de las islas son combinadas para conformar una población global de la que seleccionaremos el mejor individuo. Este enfoque tiene la ventaja de que es mucho más resistente a óptimos locales durante el proceso de búsqueda, debido a que cada población es teóricamente independiente de las demás. Una característica muy interesante de este modelo es la introducción del concepto de migración. La migración entre islas es un artefacto que proporciona el intercambio periódico de individuos, posiblemente los mejores, entre las islas de acuerdo a una topología de interconexión definida (15; 49).

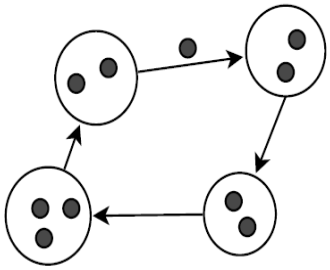


Figura 1.9 Modelo de Islas.

En el concepto de migración deben tenerse en cuenta los elementos siguientes (50; 51):

- **Política de selección de los individuos**: este proceso se realiza, en general, de dos formas: determinística y estocástica. La variante estocástica no garantiza la selección de los mejores individuos, sin embargo, su costo computacional es menor. En tanto, la variante determinística (ruleta, torneo, etc.), permite la selección de los mejores individuos. El número de individuos emigrantes puede ser un valor fijo o puede definirse la elección de un porcentaje de la población.
- **Frecuencia en que se producirán las migraciones:** representa el intervalo de tiempo que media entre las migraciones.

- **Topología de interconexión:** representa la estrategia de comunicación que seguirán las diferentes islas. Ejemplos de ellas son:
  1. **Anillo:** cada subpoblación envía sus p_i mejores individuos a una población vecina, efectuándose la migración en un único sentido de flujo.
  2. **Estrella:** existe una subpoblación que es seleccionada como maestra (aquella que tiene mejor media en el valor de la función objetivo), siendo las demás consideradas como esclavas. Todas las subpoblaciones esclavas mandan sus p_i mejores individuos ($p_i \geq 1$) a la subpoblación maestra, la cual a su vez manda sus p_j mejores individuos ($p_j \geq 1$) a cada una de las subpoblaciones esclavas.
- **Sincronización entre islas**: define la forma en que las islas se coordinan para migrar los individuos.
  1. **Modelo síncrono**: las islas se configuran para que intercambien los individuos cada cierto número de generaciones, de manera que cada isla ejecutaría de forma secuencial el algoritmo y, llegados a cierta generación preseleccionada, pasaría a la fase de migración. En la fase de migración las islas deberán sincronizarse y esperar a que se envíen y reciban los individuos. Esto provoca retardos, ya que las islas más rápidas deberán esperar por las más lentas.
  2. **Modelo asíncrono**: las fases de evolución y de migración están acopladas. Los envíos se realizan de forma asíncrona y, periódicamente, se comprueba si hay inmigrantes. De este modo, no se producen retrasos, ya que cada nodo ejecutará a la mayor velocidad posible sin tener que esperar por los nodos más lentos.

1.7. Soluciones encontradas.

Se realizó una revisión sobre las soluciones existentes con el fin de valorar la posible reutilización de algunos de sus componentes, entre ellas se encuentran:

- **MALLBA**: desarrollada en el proyecto TIC1999-0754-C03, en el que colaboran las Universidades de Málaga (MA), La Laguna (LL) y la Universidad Politécnica de Cataluña (BA). La implementación se realizó en el lenguaje de programación C++ y se desarrollaron distintas versiones de PSO (secuencial, distribuido en LAN y en WAN) (52).
- **Computational Intelligence Library** (CILib): se distribuye bajo la licencia GNU General Public License (GPL). Esta biblioteca ha sido desarrollada en Java por un grupo de investigación de la Universidad de Pretoria (53).
- **ParadisEO**: es uno de los marcos de trabajo orientado a objetos que provee los modelos paralelos y distribuidos más comunes para diferentes metaheurísticas. Este framework de código abierto

está implementado en C++, y además es multiplataforma. Se distribuye bajo la licencia CeCill[15] (54).

En el ámbito local se puede señalar el Trabajo de Diploma: "Algoritmos Paralelos para la Optimización por Nube de Partículas" de Adrián Quintero Henríquez, cuyo resultado fue una biblioteca implementada en C++ que contiene las versiones secuencial y paralela de PSO. Su principal desventaja es que fue implementada en C++ y no permite ser usada sobre la Plataforma T-arenal.

Resulta válido destacar que en ciertos problemas el desempeño del algoritmo depende en buena medida de la calidad de los números aleatorios generados (55), por este motivo se decide reutilizar ciertas clases e interfaces provenientes de la biblioteca CILib que proveen estas funcionalidades, de esta forma la biblioteca contaría con diferentes generadores de números aleatorios.

1.8. Herramientas y Tecnologías.

En este apartado se describen las herramientas y tecnologías empleadas para dar solución al problema de la presente investigación.

1.8.1. Plataforma de Tareas Distribuidas T-arenal 2.0.

La Plataforma T-arenal v2.0 es un sistema distribuido de propósito general, está implementada con una arquitectura de varios servidores para una mejor configuración y uso más equitativo de los clientes. T-arenal es un producto informático que ofrece una alternativa de cómputo y que aglutina en un solo conjunto varias estaciones de trabajo sin intentar eliminar o pretender aminorar las amplias posibilidades de aplicación de los modelos paralelos, sino de complementar todos los medios disponibles en una gran supercomputadora virtual. Esto es posible debido a que los elementos de cómputo estarán distribuidos por diferentes servidores de acuerdo a las prestaciones que cada uno tenga. La Plataforma será vista por el usuario final como un solo ordenador y todos los servidores serán gestionados a través de un servidor central (12).

[15] Licencia francesa de software libre adaptada tanto a leyes francesas como a los tratados internacionales, dentro del espíritu y compatibles con GNU GPL (71).

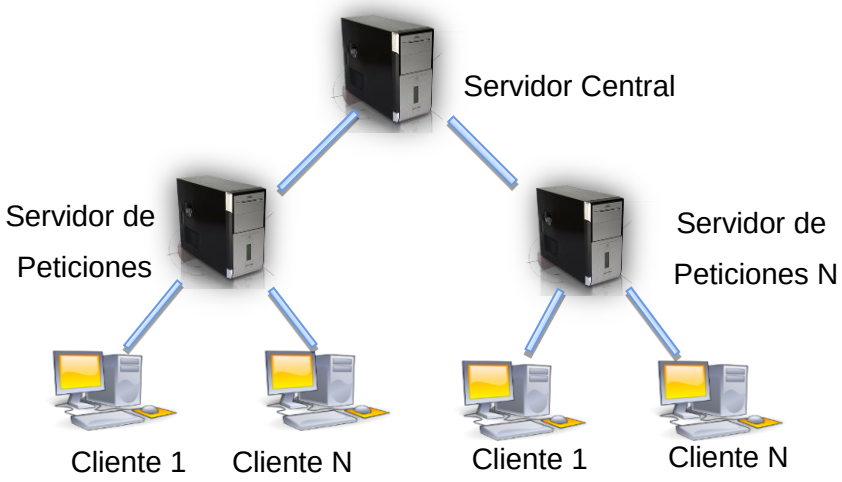


Figura 1.10 Plataforma de Tareas Distribuidas T-arenal 2.0.

1.8.2. Lenguaje de Programación.

El lenguaje de programación seleccionado es Java, la principal razón es que la plataforma T-arenal está implementada en dicho lenguaje. Entre sus características fundamentales se encuentran (56)**:**

- Orientado a objetos: Java fue diseñado como un lenguaje orientado a objetos desde el principio. Los objetos agrupan en estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manipulan esos datos.
- Portable: la independencia a la arquitectura representa sólo una parte de su portabilidad. Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas.
- Distribuido: su portabilidad permite las aplicaciones desarrolladas puedan ser ejecutadas en diferentes plataformas, lo que implica que la carga de trabajo de un sistema pueda ser distribuida entre diversos equipos de cómputo sin problemas de incompatibilidad de estructuras de datos. Esto permite aprovechar los recursos de sistemas de cómputo geográficamente separados, lo cual es la esencia del concepto de sistemas distribuidos.
- Robusto: no permite el manejo directo de memoria. Libera al usuario de preocuparse por el manejo de memoria, hace que los programas sean seguros en su ejecución, además de permitir la rápida detección de los errores en la compilación, la cual se complementa con una segunda revisión cuando el programa está en ejecución.

1.8.3. Entorno de Desarrollo Integrado (*IDE*).

Netbeans 6.9.1: es un entorno de desarrollo libre, multiplataforma, de código abierto, para el desarrollo de software. Soporta el desarrollo de todos los tipos de aplicación Java (57). Este IDE fue seleccionado principalmente por su potente editor de código y la interfaz amigable que presenta.

1.8.4. Lenguaje de Modelado.

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se emplea para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. UML incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos (58).

1.8.5. Herramienta CASE.

Ingeniería de Software Asistida por Computadoras (CASE): aplicación de métodos y técnicas enfocadas a ayudar a los analistas de software desde el inicio del ciclo de vida de un software hasta que termina con la última iteración del mismo proporcionándoles diagramas de casos de usos y de interfaces y auto generando código de programación a partir de estos diagramas (59).

Visual Paradigm para UML Edición Empresarial 8.0 (60) : herramienta de modelado visual para todos los tipos de diagrama UML. Soporta el ciclo de vida completo del desarrollo del software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue.

1.8.6. Marco de Trabajo Spring.

Spring es un marco de trabajo de código abierto creado para hacer frente a la complejidad del desarrollo de aplicaciones empresariales. Presenta un arquitectura en capas, que le permite al desarrollador decidir qué componentes utilizar, además de constituir un marco de trabajo coherente para el desarrollo de aplicaciones J2EE. Cada módulo o componente presente en el marco de trabajo Spring puede entenderse como un ente aparte, o puede usarse combinándolo con uno o varios de los restantes. Las funcionalidades de Spring pueden ser usadas en la mayoría de los servidores J2EE. El

eje central de Spring es brindar una lógica reutilizable, y objetos de acceso a datos que no estén atados a ningún servicio específico J2EE. Dichos objetos pueden ser reusados en varios entornos empresariales, aplicaciones independientes, entorno de pruebas, y muchos otros sin ningún tipo de problemas (61).

1.9. Conclusiones del capítulo.

En el presente capítulo se realizó una revisión del estado del arte de las técnicas metaheurísticas, donde se abordaron brevemente las técnicas principales. Además se describió la Optimización por Enjambre de Partículas; se realizó un análisis de sus parámetros, topologías, variantes y esquemas distribuidos. Se abordaron aspectos relacionados con la Computación Paralela y Distribuida. Finalmente se describieron las principales herramientas que se utilizarán solucionar el problema planteado. Se pudo constatar que muchos problemas para su resolución han utilizado la PSO han obtenido buenos resultados, con la distribución del método deberá reducirse el tiempo para obtenerlas.

Capítulo 2: Diseño e Implementación de la Biblioteca PSO.

En este capítulo se describen los elementos fundamentales relativos al diseño e implementación de la biblioteca. Se describen sus principales clases, interfaces y las funcionalidades que brindan cada una de ellas; patrones de diseño utilizados, así como las pautas que definen su funcionamiento básico.

2.1. Diseño de la Biblioteca PSO.

La biblioteca desarrollada es de propósito general, permitiendo resolver diferentes tipos de problemas de optimización. Para ello, cada problema debe ser adaptado de forma tal que esta pueda darle solución, con este fin, el usuario debe suministrar ciertas informaciones sobre el problema en cuestión. La utilización del paradigma orientado a objetos para la concepción de la biblioteca permitió las siguientes ventajas:

- **Reusabilidad**: una misma implementación de un algoritmo permite resolver una amplia gama de problemas muy diferentes entre sí.
- **Facilidad de ampliación y de modificación**: se dispone de varios mecanismos de ampliación como son la herencia, la redefinición de métodos, polimorfismo, que permiten ampliar las funcionalidades que brinda la biblioteca con relativa facilidad. El encapsulamiento en clases de los elementos que conforman la biblioteca permite modificar uno de ellos sin afectar al resto.
- **Facilidad de aprendizaje y transparencia de uso**: la biblioteca oculta la implementación del algoritmo, con lo que el usuario final no necesita conocerlo, únicamente rellenar las clases específicas al problema para utilizarla.

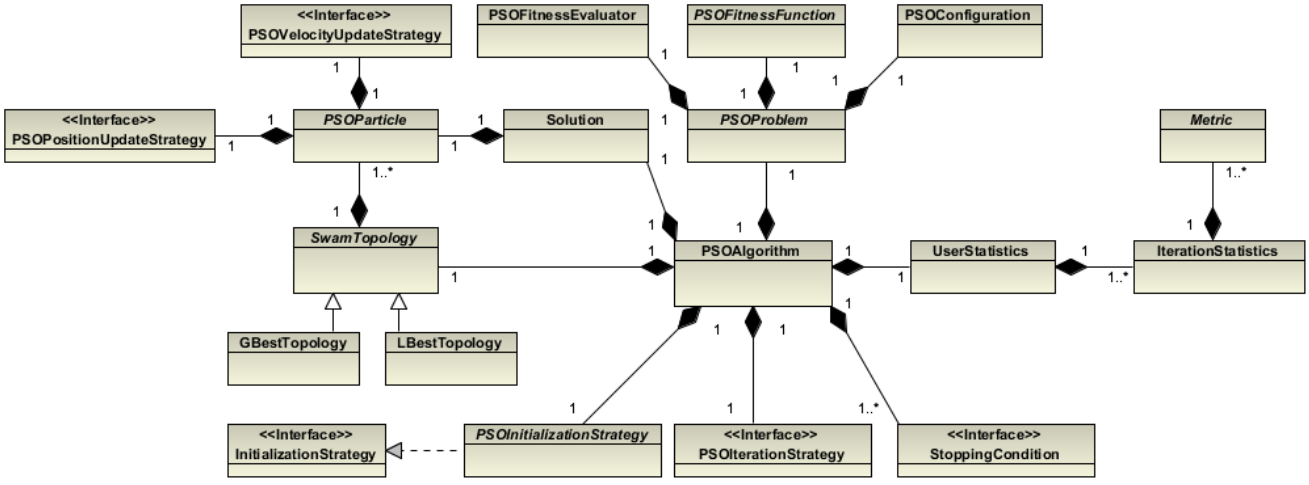


Figura 2.1 Diagrama UML de la Biblioteca PSO

Lógica interna de la Biblioteca

Una vez que el usuario haya modelado el problema (PSOProblem) que desea resolver, el algoritmo basado en PSO (PSOAlgorithm) recibe como parámetro dicho problema, de él obtiene toda la información requerida básicamente (función objetivo, configuración que utilizará el algoritmo, etc).
En este instante se encuentra apto para iniciar el proceso de resolución, en primer lugar inicializa todos los parámetros (tamaño de la población, dimensión del problema, condiciones de parada, estrategia de actualización de posición (PSOPositionUpdateStrategy) y velocidad (PSOVelocityUpdateStrategy) de las partículas, etcétera) a partir de la información provista por el problema. En este punto donde ya se han configurado todos los parámetros el usuario tiene la posibilidad de especificar las métricas que desea obtener de la ejecución, hecho esto, puede ordenar la ejecución del algoritmo. En cada iteración se evalúan las condiciones de parada y en caso de que no se cumpla alguna se ejecuta una iteración del algoritmo actualizándose la mejor solución global y demás parámetros específicos de cada problema.

Estructura de paquetes de la Biblioteca

- **pso.problem**: contiene las clases e interfaces necesarias para el modelado del problema de optimización que se desea resolver.
- **pso.particle**: incluye las clases e interfaces requeridas para representar una partícula (solución del problema que se intenta resolver), así como diferentes variantes de actualización de la velocidad.
- **pso.algorithm:** define las clases e interfaces que permiten resolver el problema de optimización (estrategias de confinamiento que se le aplicarán a las partículas, condiciones de parada del algoritmo, etc.).
- **pso.initializationstrategies**: aglutina las clases e interfaces que permiten construir el enjambre inicial de partículas.
- **pso.topologies**: reúne las clases que posibilitan definir las topologías del enjambre de partículas.
- **pso.migrationstrategies**: incluye las clases e interfaces para realizar la migración de partículas entre subenjambres.
- **pso.io**: contiene las clases que permiten realizar operaciones de entrada y salida (carga y escritura de datos de ficheros).
- **pso.util**: define las clases e interfaces para ejecutar funciones auxiliares (generación de números aleatorios, operaciones con vectores, etc).

- **pso.parameters**: agrupa las clases e interfaces que representan los diferentes parámetros (factor inercia, coeficientes de aprendizaje, dimensión del problema, etc.) requeridos para resolver un problema con PSO.

2.2. Patrones de diseño utilizados.

Un patrón es una solución de un problema en determinado contexto, un contexto es una situación en la que el patrón es aplicable y que generalmente es recurrente. A continuación se describen los patrones de diseños más importantes aplicados en el desarrollo de la biblioteca.

Estrategia

- **Problema**: el enjambre de partículas puede adquirir diversas configuraciones o topologías de acuerdo al problema específico que se intenta resolver, por lo que nunca se podrá conocer a priori cuál es la topología adecuada para un problema específico.
- **Solución**: se diseña una clase abstracta SwarmTopology que implementa la lógica de la configuración de un enjambre de partículas. De esta forma el algoritmo carga dinámicamente la clase que describe este comportamiento, obviando completamente la naturaleza del problema que se desea resolver.

Inyección de dependencias

- **Problema**: se necesitaba un bajo acoplamiento entre el problema y la estrategia de inicialización del enjambre de partículas.
- **Solución**: se definió una interfaz InitializationStrategy con las funcionalidades que debe tener el inicializador del enjambre de partículas. La PSOProblem tiene una referencia a un objeto de tipo InitializationStrategy, pero no conoce hasta tiempo de ejecución quien será la clase concreta (RandomBoundedInitializationStrategy) que implementó la interfaz. Este diseño permite que se puedan adicionar y cambiar de forma dinámica algoritmos de inicialización del enjambre de partículas.

2.3. Implementación de la Biblioteca de PSO.

En este apartado se describen las clases, interfaces y métodos fundamentales de la biblioteca.

2.3.1. Principales Clases de la Biblioteca.

Clase PSOProblem

Esta clase representa el problema de optimización que se intenta resolver, debe contener los elementos básicos que definen el problema específico, si el problema así lo requiere, debe permitir leer estos datos desde un fichero donde se indique la instancia y la configuración que utilizará el algoritmo para su ejecución.

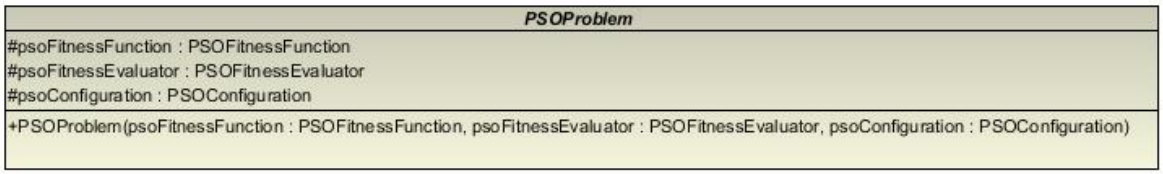


Figura 2.2: UML de la clase *PSOProblem.*

Clase PSOParticle

Un elemento fundamental en la definición de un problema de optimización para ser resuelto con metaheurísticas, lo constituye la representación de sus soluciones (62), en el caso de PSO son las partículas que formarán el enjambre. La clase PSOParticle contiene los elementos básicos de una partícula: vectores de posición y velocidad, valor de fitness, etc. El método fundamental de esta clase es **fly**(), el cual actualiza los vectores de posición y velocidad de la partícula, es decir, mueve la partícula hacia otra posición en el espacio de búsqueda.

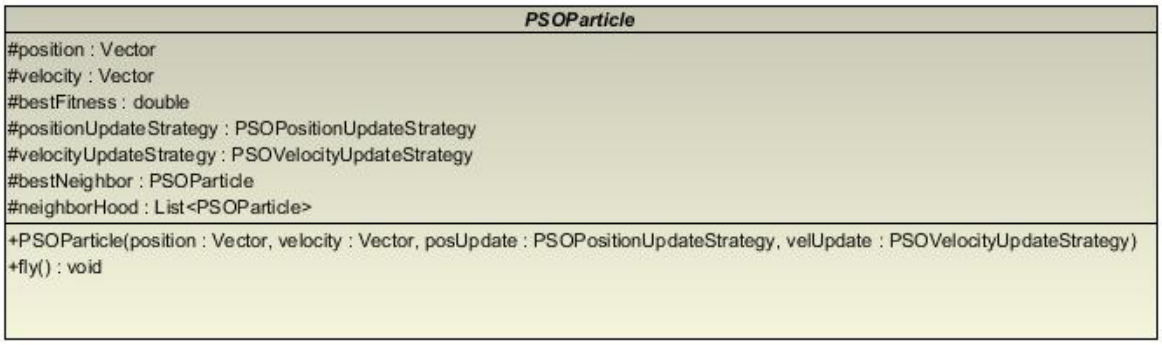


Figura 2.3: UML de la clase *PSOParticle.*

Clase PSOFitnessFunction

Su principal objetivo es cuantificar la calidad de las soluciones encontradas ofreciendo una guía para realizar el proceso de exploración del espacio de soluciones (62), para ello, se definió el método abstracto calculateFitness, que recibe como parámetro una solución (partícula) y devuelve un valor real que brinda una medida de la bondad de la solución encontrada por la partícula en ese instante.

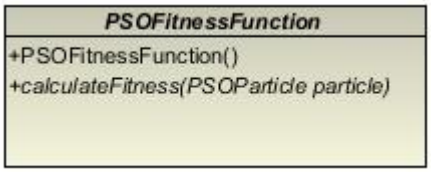


Figura 2.4: UML de la clase *PSOFitnessFunction.*

Clase PSOConfiguration

Esta clase contiene los parámetros necesarios para la ejecución del algoritmo los cuales dependerán del problema que se desee resolver, por ejemplo: dimensión del espacio de búsqueda, topología del enjambre de partículas, estrategia de inicialización de las partículas, política de actualización de la posición o velocidad de la partícula, origen de la instancia del problema que se desea resolver, etc.

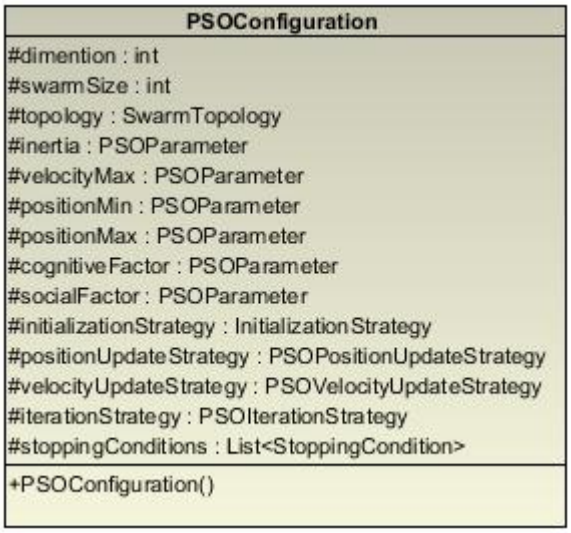


Figura 2.5: UML de la clase *PSOConfiguration.*

Clase SwarmTopology

Define la configuración del enjambre de partículas, su objetivo es definir cómo estarán relacionadas las partículas entre sí. El método fundamental de esta clase es ***setUpTopology()*** el cual permite definir una topología específica para el enjambre. La biblioteca tiene implementadas la *GBestTopology* o topología global (la vecindad de la partícula es todo el enjambre) y la *LBestTopology* o topología local (la partícula tiene un vecindario de tamaño k), en ambos casos se incluye la partícula en su vecindad.

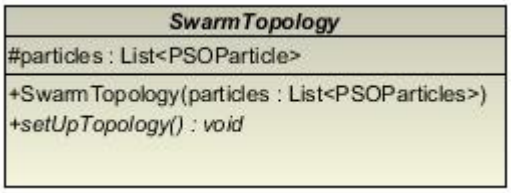


Figura 2.6 UML de la clase *SwarmTopology.*

Clase PSOAlgorithm

Representa el algoritmo basado en PSO que intentará encontrar una solución para el problema planteado. Esta clase contiene las funcionalidades necesarias para la ejecución secuencial del algoritmo PSO.

Sus métodos fundamentales son:

- void ***initialize()***: inicializa el enjambre de partículas y configura todos los parámetros necesarios para la correcta ejecución del algoritmo.
- void ***run()***: ejecuta el algoritmo basado en PSO hasta que se alcance algún criterio de parada especificado con anterioridad por el usuario.
- void ***performIteration()***: ejecuta una iteración del algoritmo PSO, según se haya definido en la estrategia de iteración (iterationStrategy).
- boolean ***isFinished()***: determina si se ha alcanzado alguna condición de parada que indique que se debe detener la ejecución.
- Solution ***getBestSolution()***: devuelve la mejor solución encontrada.

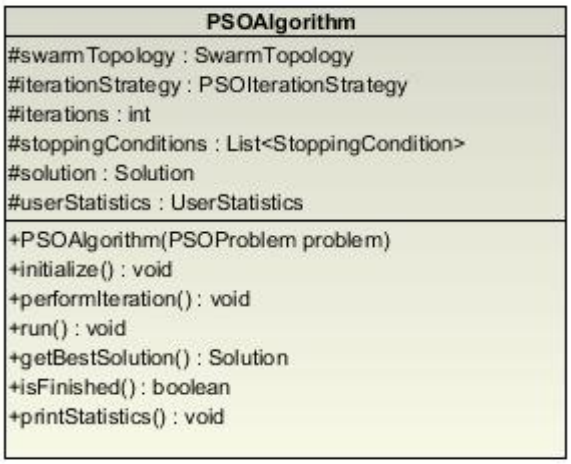


Figura 2.7 UML de la clase *PSOAlgorithm*

Clase UserStatistics

Esta clase tiene como objetivo administrar los datos relevantes desde el punto de vista estadístico para el usuario acerca de la evolución del algoritmo a lo largo del proceso de resolución. Su propósito fundamental es la definición de las métricas se van a recolectar: por ejemplo: mejor fitness encontrado, promedio de fitness del enjambre de partículas, tiempo de ejecución, etc.

Sus métodos fundamentales son:

- *void* ***addMetric****(Metric metric)*: adiciona una métrica especificada por el usuario.
- void ***collectMetrics****(PSOAlgorithm psoAlgorithm)*: recolecta la información de interés (métricas) para el usuario.
- void ***printStatistics()*:** imprime por consola la información recopilada.
- void ***saveToFile***(*File file*): almacena en un fichero la información obtenida.

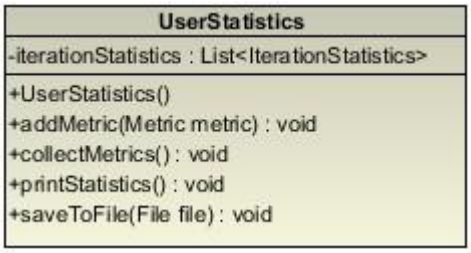


Figura 2.8 UML de la clase *UserStatistics*

2.3.2. Principales Interfaces de la Biblioteca.

PSOVelocityUpdateStrategy

Define cómo se actualizará la velocidad de cada partícula del enjambre según la variante que se vaya a utilizar. Por ejemplo: con inercia decreciente, con factor de constricción, etc.
Sus métodos fundamentales son:

- *void* ***updateVelocity****(PSOParticle particle):* actualiza la velocidad de la partícula que recibe como parámetro.
- *void* ***updateParameters****(PSOAlgorithm psoAlgorithm):* actualiza, si así lo requiere la variante implementada, los parámetros de la misma según el estado de resolución del algoritmo.

La biblioteca brinda al usuario la posibilidad de seleccionar diferentes variantes para efectuar la actualización de la velocidad de las partículas.

PSOPositionUpdateStrategy

Define cómo se actualizará la posición de las partículas del cúmulo según la variante que desee utilizar. Para ello el usuario debe implementar el método:

- void ***updatePosition****(PSOParticle particle):* actualiza la posición de la partícula que recibe como parámetro.

StoppingCondition

Representa una condición que indica que se debe detener la ejecución del algoritmo. Las más comunes son: se alcanzó un número de iteraciones predefinido, se obtuvo una solución con una calidad aceptable, estancamiento en el proceso de optimización, etc. Para ello el usuario debe implementar los métodos:

- boolean ***evaluateCondition***(*PSOAlgorithm psoAlgorithm*): devuelve verdadero si se cumple la condición de parada para el algoritmo, falso en caso contrario.
- double ***getPercentageCompleted***(*PSOAlgorithm psoAlgorithm*): devuelve un valor real que representa el porcentaje de completamiento del algoritmo según la condición de parada.

PSOMigrationStrategy

Define cómo se realizará la migración de las partículas de los subenjambres según la topología de interconexión seleccionada.

2.4. Integración con la Plataforma T-arenal.

Concluido el modelado el problema a resolver con la biblioteca, el usuario debe heredar de dos clases: DataManager y Task, para lograr una ejecución distribuida del algoritmo. Se deben redefinir determinados métodos a fin de establecer su aplicación distribuida (63).

- **Clase DataManager.**

Esta clase se ejecuta en un servidor de peticiones determinado. Su propósito es generar las unidades de trabajo que se enviarán a los clientes, procesar los resultados devueltos por los clientes, ajustar la granularidad (tamaño) de las unidades de trabajo, generar la información sobre el estado en que se encuentra la tarea en un instante determinado así como analizar el cómputo distribuido.

- **Clase Task.**

Esta clase supone una instancia del algoritmo PSO, el cual se ejecutará de forma secuencial en cada uno de los clientes. Para ello, debe implementar el método processUnit que recibe como parámetro un vector con los datos del problema que se desea resolver y una vez ejecutado el algoritmo, este debe devolver igualmente un vector que contendrá, posiblemente, la mejor(es) solución(es) encontrada(s).

2.5. Conclusiones del capítulo.

En este capítulo se expuso la propuesta de diseño e implementación de la biblioteca, se explicaron las principales funcionalidades que brindan sus principales clases e interfaces para dar solución al problema de la presente investigación. Finalmente se describió brevemente cómo se debe realizar la integración de la biblioteca con la Plataforma T-arenal.

Capítulo 3: Resultados.

En este capítulo se presentan las variantes que contiene la biblioteca. Además se exponen y discuten los experimentos y resultados obtenidos de la aplicación de la biblioteca en la Plataforma de Tareas Distribuidas T-arenal en la resolución del *K-Means Problem*.

3.1. Variantes incluidas en la Biblioteca.

Se desarrolló una biblioteca que incluye diferentes variantes de la técnica seleccionada: PSO con peso inercial decreciente (33), con factor de constricción (34), Enjambre de Partículas Totalmente Informado (FIPS) (35), PSO Estándar 2006 y 2007 (55) y PSO disipativo (64).

3.2. Caso de estudio: *K-Means Clustering Problem*.

En este acápite se realiza una breve introducción al problema del *clustering*, se exponen sus aplicaciones en la práctica. Además se define formalmente el problema que será resuelto con la biblioteca, se explica su complejidad y se presentan algunos índices empleados para evaluar los resultados obtenidos.

Introducción al Clustering

El proceso de agrupar un conjunto de objetos abstractos o físicos en clases similares recibe el nombre de *clustering*. Un *cluster*[16] es una colección de datos que son parecidos entre ellos y diferentes a los datos pertenecientes a otros *clusters*. El problema del *clustering* ha sido abordado por gran cantidad de disciplinas y es aplicable a una gran cantidad de contextos, lo cual refleja su utilidad como uno de los pasos en el análisis experimental de datos. En el área de los negocios, el *clustering* puede ayudar a descubrir distintos grupos en los hábitos de sus clientes y así, caracterizarlos en grupos basados en patrones de compra. En el ámbito de la biología puede utilizarse, por ejemplo, para derivar taxonomías de animales y vegetales o descubrir genes con funcionalidades similares. El análisis de *clusters* se puede usar para hacerse una idea de la distribución de los datos, para observar las características de cada *cluster* y para centrarse en un conjunto particular de datos para futuros análisis (64; 65).

[16]Del inglés significa racimo, conjunto o cúmulo.

3.2.1. Definición del problema

Dado un conjunto de m puntos $P = \{p_1, p_2, \dots, p_m\}$ definidos en $\mathbb{R}^n$, encontrar un conjunto de *k* puntos $C = \{c_1, c_2, \dots, c_k\} \subset \mathbb{R}^n$ tal que: $\sum_{j=1}^{m} dist(p_j, C)$ sea mínima, donde: dist (p_j,C) es la distancia del punto p_j al punto más cercano del conjunto C y n es la dimensión de cada punto del conjunto de datos, donde: $1 \leq k \leq m$, (64; 66). Se define la función $f: \mathbb{R}^n \to \mathbb{R}$ como $f(\boldsymbol{C}) = \frac{\sum_{j=1}^{k}\left[\sum_{z \in C_j} \boldsymbol{d}(z, c_j)\right]}{\boldsymbol{m}}$, donde z representa un punto del conjunto de datos y $\boldsymbol{c_j}$ representa el j-ésimo *cluster* y $\boldsymbol{d}$ es la distancia euclídea.

Complejidad del problema

La complejidad de este problema está bien estudiada, como se muestra en (66), pertenece a la clase de los NP-duros, a pesar de que la complejidad de la función objetivo de este problema es de orden lineal en función de la cantidad de puntos *O(knm)* el tamaño del espacio de búsqueda asociado a este problema es exponencial lo que provoca que la búsqueda de la solución óptima sea intratable computacionalmente.

Métricas de evaluación del *clustering*.

1. Distancia Intra-Cluster (Cohesión): determina lo cercanos que están los objetos dentro del cluster.

$$IntraC = \sum_{i=1}^{K}\left(\sum_{x,t \in C_i} dist(x, ci)/m_i\right)$$

donde: C_i representa el *cluster i*, *K* es la cantidad de *clusters* y m_i es la cantidad de elementos pertenecientes al cluster *i*.

2. Distancia Inter-Cluster (Aislamiento): determina lo distinto y bien separado que está un *cluster* con respecto a los otros.

$$InterC = \frac{\sum_{i=1}^{K-1}\sum_{j=i+1}^{K} dist(c_i, c_j)}{\sum_{i=1}^{K-1} i}$$

donde: ci representa al elemento central (centroide) del *cluster i*, y c_j representa el centroide del cluster *j* y *K* es la cantidad de *clusters*.

3.3. Modelado del problema.

Es posible ver al problema del *clustering* como un problema de optimización que localiza los centroides óptimos de los clusters. En este contexto, una partícula representa los k centros de los *clusters*. Es decir, que cada partícula (pi) se construye de la siguiente manera: $p_i = (c_1,c_2,\ldots,c_k)$, donde c_k representa el centro del *cluster* k. Para manejar los posibles *clusters* vacíos durante la optimización se determinó usar una estrategia de penalización en la función objetivo.

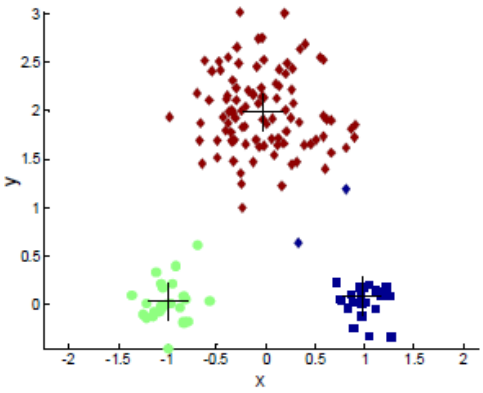


Figura 3.1 Representación de una partícula con 3 centros.

3.4. Resolución del problema.

Para resolver el *K-Means* se utilizó la versión de PSO para espacio de búsqueda continuo y se definieron e implementaron clases para este problema específico. A continuación se señalarán las clases implementadas.

- **ClusteringParticle**: representa el vector de centros (ver sección 3.3. Modelado del problema.).
- **ClusteringInitializationStrategy**: estrategia de inicialización del enjambre de partículas.
- **ClusteringFitnessFunction**: función de aptitud (ver sección 3.2.1. Definición del problema).
- **Cluster**: *cluster* del conjunto de datos.
- **ClusteringUtils**: realiza las operaciones con el conjunto de puntos (cálculo de distancias, asignación de puntos a clusters).

3.5. Experimentos y Resultados.

En este apartado se describen los experimentos realizados y se exponen los resultados obtenidos en el proceso de resolución del caso de estudio con la biblioteca.

- **Descripción de las instancias.**

Nombre	Dimensión	Cantidad de Clusters	Cantidad de Puntos
Iris-Plants	4	3	150
6300D50K100	50	100	6300
12500D100K100	50	100	12500
25000D100K100	50	100	25000

Se utilizaron 4 instancias (1 real y 3 artificiales). La instancia real corresponde a “Iris Plants” (base de datos de 4 atributos numéricos, 3 clases y 150 elementos). El resto de las instancias fueron generadas empleando el software Weka 3.6 siguiendo un patrón aleatorio para la generación de los conjuntos de datos. Todas las instancias tienen igual cantidad de puntos por cluster.

3.5.1. Experimentos secuenciales.

Descripción de las pruebas

Las pruebas fueron realizadas en una PC con procesador Intel (R) Pentium (R) 4 de 3.00 Ghz y 1GB de memoria RAM. Con el propósito de comprobar la correctitud del algoritmo propuesto se realizaron 10 ejecuciones independientes con la instancia *Iris-Plants* empleando un enjambre con topología global con 10 partículas, para actualizar las velocidades de las partículas se emplearon los siguientes parámetros (c_1=c_2=1.193, w=0.721) empleando 100 iteraciones. El enjambre inicial se construyó de forma aleatoria escogiendo puntos pertenecientes al conjunto de datos.

Resultados obtenidos

Instancia	Distancia IntraCluster	Distancia InterCluster
Iris-Plants	0.99	3.24

Tabla 3.1 Resultados promedios obtenidos para la distancias intracluster e intercluster.

Instancia	Distancia IntraCluster	Distancia InterCluster
Iris-Plants	0.93	3.16

Tabla 3.2 Resultados promedios obtenidos para la distancias intracluster e intercluster en (67)

Al comparar los resultados alcanzados con la biblioteca con los obtenidos por Villaga (67) no se aprecian diferencias significativas respecto a estas métricas e incluso la propuesta alcanza mejores

resultados en términos de distancia Inter-Cluster. Se calcularon además las distancias de los elementos de un *cluster* con respecto a su centro y el mejor resultado obtenido (96.74) es comparable con resultados obtenidos por (Yang,Hsiao y Chuang) (68). Estos resultados parciales a pesar de ser una instancia pequeña permiten corroborar la validez en el modelado y resolución del problema.

Para el resto de las instancias se utilizaron los siguientes parámetros:

- ***Topología***: Global Best.
- ***Cantidad de partículas***: 100.
- ***Cantidad de evaluaciones de la función objetivo***: 1000.
- ***Factor inercia***: 0.72.
- ***Coeficiente cognitivo:*** 1.193.
- ***Coeficiente social:*** 1.193.

El enjambre inicial se generó de la siguiente forma: el 80% de la población se seleccionaron puntos aleatorios del conjunto de datos y el 20% restante de forma aleatoria con distribución uniforme.
Se realizó una ejecución secuencial por cada instancia y como se esperaba al aumentar el tamaño del conjunto de datos se produce un incremento importante del tiempo de ejecución, llegando a superar las 5 horas en el caso de la instancia de mayor tamaño.

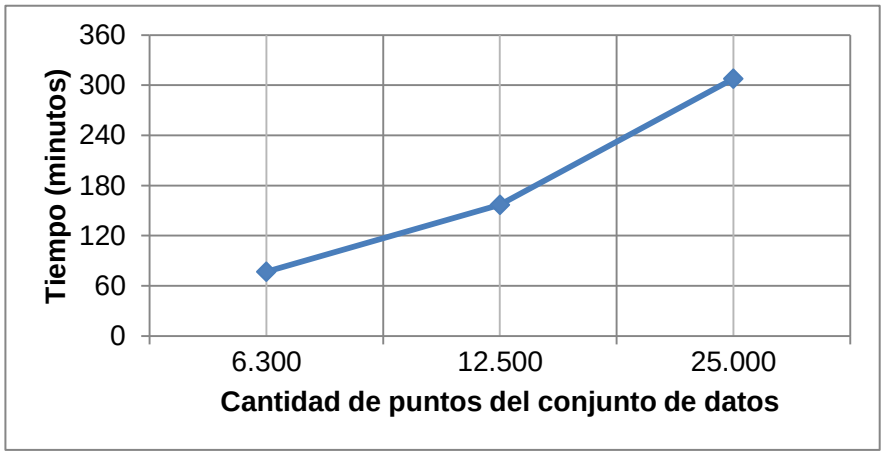


Figura 3.2 Tiempo de ejecución del algoritmo secuencial.

3.5.2. Experimentos distribuidos.

En esta sección se exponen y discuten los resultados alcanzados mediante una aproximación distribuida al problema.

Descripción de los experimentos

Los experimentos fueron realizados en el Laboratorio 402 donde se encuentra desplegada la Plataforma T-arenal. El Laboratorio cuenta con 30 PCs de prestaciones similares a la empleada en las pruebas secuenciales y una red con tasa de transferencia de 100Mbps.Resulta importante señalar que estas pruebas se efectuaron en el horario de la noche y madrugada.

El modelo de distribución seleccionado es el modelo de islas síncrono con comunicación en anillo, la migración se efectuó cada 10 iteraciones y se migró el 10% de los individuos. Por cuestiones de disponibilidad de tiempo este proceso se realizó hasta completar 1000 evaluaciones con la misma configuración utilizada en el experimento secuencial.

Resultados obtenidos

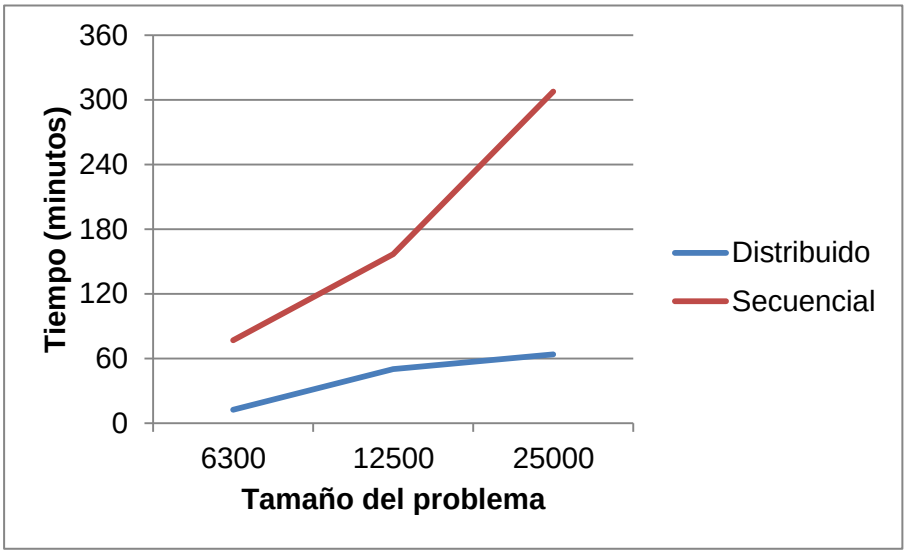


Figura 3.3 Comparación entre tiempo secuencial y distribuido en función del tamaño del problema.

En la (figura 3.3) se evidencia el comportamiento de los tiempos secuenciales y distribuidos variando el tamaño del problema. Se observa cómo la curva que describe el tiempo secuencial se mantiene muy por encima de la curva para el esquema distribuido. Nótese que al incrementarse casi de forma lineal el tiempo secuencial, el aumento del tiempo distribuido no es significativo.

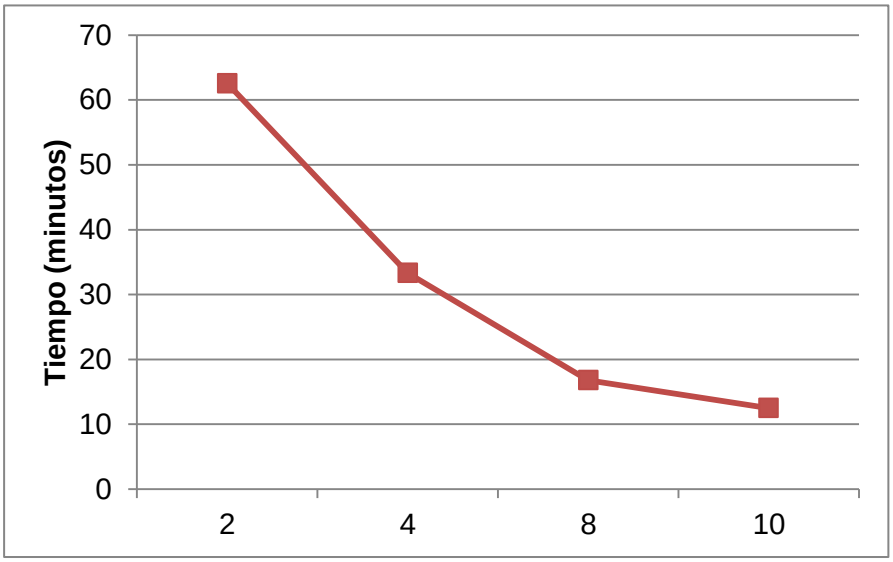


Figura 3.4 Reducción del tiempo para la instancia 6300D50K100.

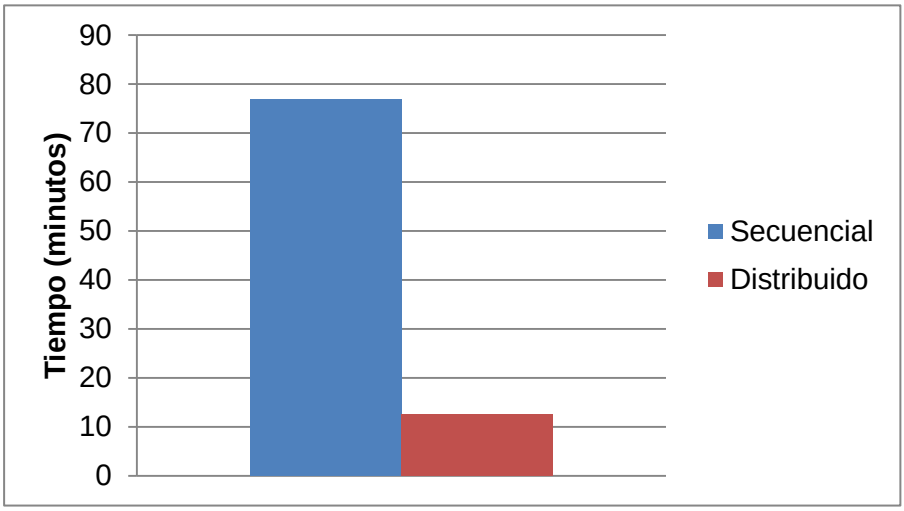
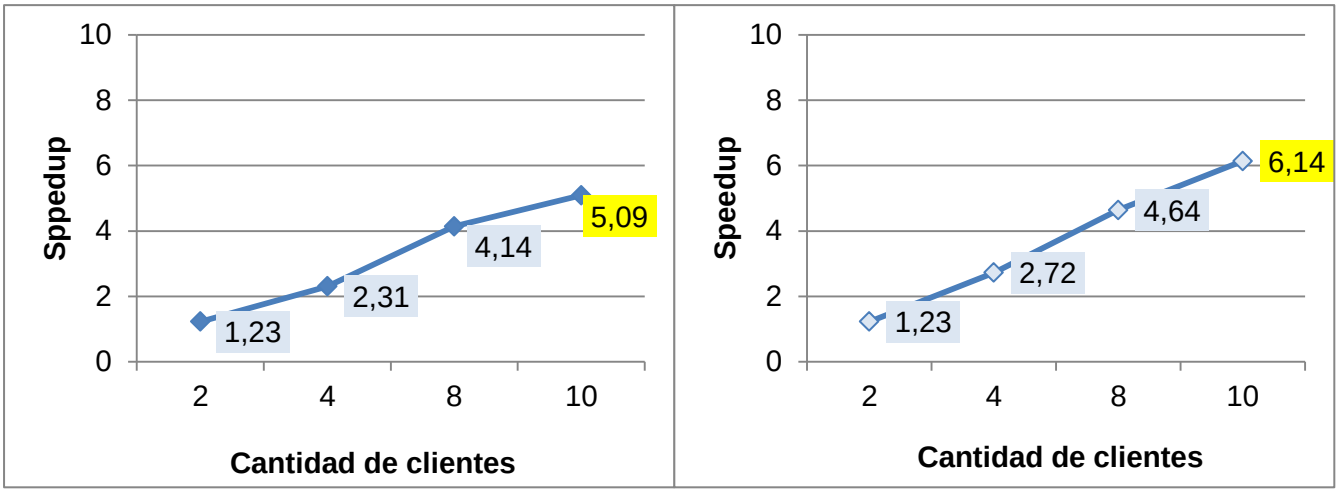


Figura 3.5 Comparación de tiempo secuencial y distribuido empleando 10 clientes para la instancia 6300K50100.

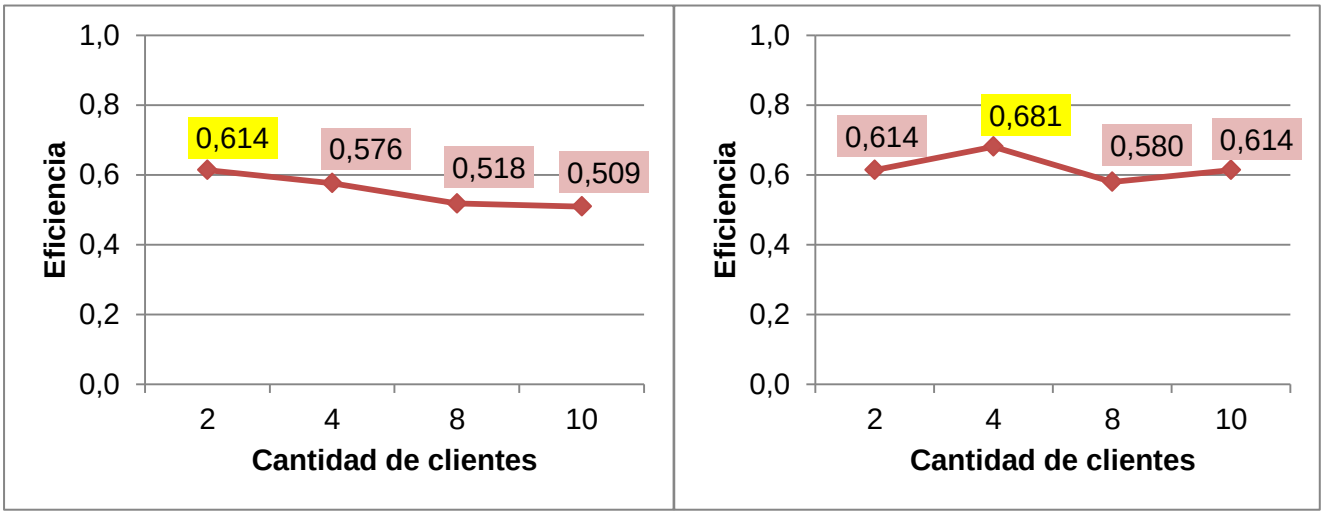
En las (figuras 3.4 y 3.5) se muestra la reducción del tiempo de ejecución del algoritmo al realizarlo de forma distribuida, en el caso de la (figura 3.4) ,visiblemente al aumentar la cantidad de clientes se reduce considerablemente el tiempo de ejecución (de 62 minutos con 2 clientes a 12 minutos con 10 clientes). La figura 3.5 muestra el mejor desempeño alcanzado en cuanto a tiempo, se evidencia una notable reducción de este indicador con respecto al algoritmo secuencial empleando 10 clientes (de 76 minutos (secuencial) a 12 minutos (distribuido) aproximadamente).



a) Speedup promedio b) Speedup máximo

Figura 3.6 Speed-up en función de la cantidad de clientes.

En la figura 3.6 se muestra el desempeño de la propuesta en términos de ganancia de velocidad, en la figura (3.6a)) se aprecia el comportamiento promedio de esta métrica, es decir, lo que normalmente ocurre en el sistema, se evidencia un aumento de la ganancia de velocidad en la medida en que se incrementa la cantidad de clientes. La figura (3.6b)) muestra el mejor desempeño alcanzado el cual refleja un crecimiento importante del speed-up logrando reducir los tiempos hasta 6 veces (de 76 a 12 minutos aproximadamente) utilizando para ello 10 procesadores. Los mejores desempeños alcanzados oscilan entre 1,23 con 2 clientes y 6.14 con 10 clientes.



a) Eficiencia promedio b) Eficiencia máxima

Figura 3.7 Eficiencia en función de la cantidad de procesadores.

La figura 3.7 muestra el comportamiento de la eficiencia de la propuesta, este indicador mide el grado de aprovechamiento de los clientes en la resolución del problema, en entornos distribuidos esta métrica depende de varios factores como por ejemplo: la disponibilidad de las estaciones de trabajo ya que las mismas son de tiempo compartido. Se aprecia en los resultados promedios (figura 3.7 a)) una tendencia a disminuir ligeramente la eficiencia en tanto se incrementa la cantidad de clientes, los valores de la misma oscilan entre 0.509 y 0.614, alcanzando el mejor resultado (0.614) con 2 clientes. Para el mejor desempeño alcanzado (figura 3.7 b) este parámetro registra una tendencia a mantenerse constante aun cuando se observan ciertas variaciones, las cuales están relacionadas con el entorno en que se realizan las pruebas. Este indicador fluctúa entre 0.58 y 0.68, siendo este último el valor más alto alcanzado empleando 4 clientes.

3.6. Conclusiones del capítulo.

En este capítulo se presentaron los principales resultados de la aplicación de la biblioteca al *K-Means*, los mismos demuestran la eficiencia de la solución en el caso de estudio propuesto. Se demostró que la aproximación distribuida del algoritmo es capaz de reducir el tiempo necesario para obtener una solución al problema planteado, logrando reducir en promedio hasta 5 veces el tiempo en relación al esquema secuencial empleando 10 clientes. Se logró disminuir el tiempo para determinar una solución para el caso de estudio planteado con buenos índices de prestaciones en términos de ganancia de velocidad y eficiencia.

Conclusiones

En la presente investigación se realizó un minucioso estudio de la Optimización por Enjambre de Partículas. Se desarrolló una biblioteca que permite resolver de forma secuencial y distribuida sobre la Plataforma de Tareas Distribuidas T-arenal problemas de optimización que demanden intenso cómputo, en la misma se incluyen diferentes variantes de la PSO. Las pruebas de rendimiento realizadas a la propuesta evidenciaron el buen desempeño en términos de ganancia de velocidad y eficiencia del modelo de distribución seleccionado para el caso de estudio propuesto. Como hipótesis investigativa de este trabajo se planteó que empleando algoritmos basados en la técnica de Optimización por Enjambre de Partículas distribuida sobre la Plataforma T-arenal era posible disminuir el tiempo necesario para resolver problemas de optimización de elevado costo computacional. Los resultados alcanzados permiten afirmar que se ratifica la hipótesis propuesta en la presente investigación.

Recomendaciones

Con el desarrollo de esta investigación quedaron abiertos ciertos aspectos que merecen ser tratados posteriormente, en tal sentido se propone:

- Aplicar la biblioteca a otros problemas de optimización.
- Extender la funcionalidad de la biblioteca a partir de la inclusión en la misma de nuevas variantes.
- Analizar otros modelos de distribución y valorar su posible implementación sobre la Plataforma T-arenal.

Bibliografía

1. **González, Federico Díaz.** Notas del Idioma. [En línea] 17 de Julio de 2008. [Citado el: 10 de Noviembre de 2011.] http://notasdelidioma.blogspot.com/2008/07/notas-del-idioma-65.html.

2. **Rasúa, Rafael Arturo Trujillo.** *Algoritmos paralelos para la solución de problemas discretos aplicados a la decodificación de señales.* Valencia : s.n., 2009. Tesis doctoral.

3. **Glover, Fred.** *Tabu search: Part I.* 1989.

4. *Optimization by simulated annealing.* **Kirkpatrick, S., Gelatt, C. D y Vecchi., M.P.** 1983, Science, págs. 671-680.

5. *Adaptation in Natural and Artificial Systems.* **Holland, John Henry.** 1975.

6. *Ant colony optimization theory: A survey.* **Blum, Cristian y Dorigo, Marco.** 2005 : s.n.

7. **Aguilera Mendoza, Longendri.** *Sistema de cómputo distribuido aplicado a la Bioinformática.* La Habana : s.n., 2008. Tesis de maestría.

8. **Alba, Enrique.** *Parallel Metaheuristics: A New Class of Algorithms.* s.l. : John Wiley & Sons, 2005.

9. *Particle swarm optimization.* **Kennedy, James y Eberhart, Russell.** 1995. págs. 1942-1948. Proceedings of IEEE International Conference on Neural Networks.

10. **Nieto, José Manuel García.** *Algoritmos Basados en Cúmulos de Partículas Para la Resolución de Problemas Complejos.* 2006. Tesis.

11. **Byung-Il , Koh, George, Alan D. y Haftka, Raphael.** *Parallel asynchronous particle swarm optimization.* s.l. : International Journal for Numerical Method in Eingneering, 2006.

12. **Jacas, César Raúl García.** *Manual del Usuario.Front-end de la Plataforma de Tareas Distribuidas.Interfaz de Escritorio.T-arenalVersión 2.0.* La Habana : s.n., 2010.

13. **Almaguer, Dannier Trinchet.** *Algoritmo paralelo para el modelado de yacimientos lateríticos.* 2010. Tesis de maestría.

14. **Kelley, C.T.** *Iterative Methods for Optimization.* North Carolina : s.n., 1999.

15. **Talbi, El-Ghazali.** *Metaheuristics from design to implementation.* New Jersey : Addison Welsley&Sons,Inc.,Publications, 2009.

16. *A Heuristic Approach for the Travelling Tournament using Optimal Travelling Salesman Tours.* **Ryckbosch, Frederick , Vanden Berghe, Greet y Kendall, Graham.** 2008. n Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling.

17. *Future paths for integer programming and links to artificial intelligence.* **Glover, Fred.** Mayo de 1986, Computers and Operation Research, Vol. 13.

18. *Metaheuristics in combinatorial optimization: Overview and conceptual comparison.* **Blum, Christian y Roli, Andrea.** New York : s.n., 2003, ACM Computing Surveys, Vol. 35, págs. 268-308.

19. **Glover, Fred y Kochenberger, Gary.** *Handbook of Metaheuristics.* s.l. : Kluwer Academic Publishers, 2002.

20. *Greedy Randomized Adaptive Search Procedures.* **Feo, T. A. y Resende, M. G. C.** 1999. Journal of Global Optimization.

21. *Variable neighborhood search. .* **M., Mladenovic y Hansen, P.** s.l. : Computers and Operations, 1997.

22. *Efficient local search with search space smoothing: A case study of the traveling salesman problem.* **Gu, J. y Huang, X.** 1994. IEEE Transactions on Systems Man and Cybernetics.

23. *Guided local search.* **Voudouris, C. y Tsang, E.** 1999. European Journal of Operational Research.

24. *Outline for a logical theory of adaptive systems.* **Holland, John Henry.** 1962.

25. *Heuristics for integer programming using surrogate constraints.* **Glover, Fred.** 1977.

26. No Free Lunch Theorems. [En línea] [Citado el: 6 de Junio de 2012.] http://www.no-free-lunch.org/.

27. **Reynolds, Craig.** *Flocks, herds and schools: a distributed behavioral model.* s.l. : Computer Graphics, 1987.

28. **Wilson, Edward.** *Sociobiology: The new synthesis.* s.l. : Belknap Press,Cambridge,MA, 1975.

29. **López, Jesús Ramón Pérez.** *Contribución a los métodos de optimización basados en procesos naturales y su aplicación a la medida de antenas en campo próximo.* Santander : s.n., 2005. Tesis doctoral.

30. **Kennedy, James y Eberhart, Russell.** *Swarm intelligence.* San Francisco : Morgan Kaufmann Publishers, 2001.

31. **Clerc, Maurice.** *Initialisations for Particle Swarm Optimisation.* 2008.

32. *Sampling with Hammersley and Halton Points.* **Wong, Tien-Tsin, Luk, Wai-Shing y Heng, Pheng-Ann.** 1997.

33. *Empirical Study of Particle Swarm Optimization.* **Shi, Yuhui y Eberhart, Russell.** 1999. In Proceedings of the 1999 IEEE Congress on Evolutionary Computation.

34. *The particle swarm: Explosion, stability, and convergence in a multidimensional complex space.* **Kennedy, James y Clerc, Maurice.** 2002. IEEE Transactions on Evolutionary Computation.

35. **Mendes, Rui, Kennedy, James y Neves, José.** *The Fully Informed Particle Swarm: Simpler, Maybe Better.* s.l. : IEEE TRANSACTIONS OF EVOLUTIONARY COMPUTATION, 2004. Vol. 8.

36. **Kennedy, James.** *Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance.* 1999. págs. 1931-1938. Vol. 3.

37. **Kennedy, James y Mendes, Rui.** *Neighborhood Topologies in Fully Informed and Best-of-Neighborhood Particle Swarms.* 2006. págs. 515-519.

38. *Particle swarm optimiser with neighbourhood operator.* **Suganthan, Ponnuthurai Nagaratnam .** Washington : s.n., 1999. Vol. 3, págs. 1958-1962.

39. *Particle Swarm Optimization, Methods, Taxonomy and Applications.* **Sedighizadeh, Davoud y Masehian, Ellips.** 5, Diciembre de 2009, International Journal of Computer Theory and Engineering, Vol. 1. 1793-8201.

40. *A discrete binary version of the particle swarm algorithm.* **Kennedy, James y Eberhart, Russell.** Orlando : IEEE Conference Publications, 1997. Vol. 5. Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics.

41. **Foster, Ian.** *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering.* Boston : Addison-Wesley Longman Publishing Co., 1995.

42. *Some Computer Organizations and Their Effectiveness.* **Flynn, Michael.** 9, s.l. : IEEE Transactions on Computers, 1972.

43. **Wu, Jie.** *Distributed System Design.* s.l. : CRC-Press, 1998.

44. **Tanenbaum, Andrew. y Steen, Marten Van.** *Distributed Systems: Principles and Paradigms.* s.l. : Prentice Hall, 2002.

45. *Seti@ home-massively distributed computing for seti.* **Korpela, E., y otros, y otros.** s.l. : Computing in Science and Engineering, 2001.

46. *Do-it-yourself climate prediction.* **Allen, M.** 1999.

47. *Folding@ home and genome@ home: Using distributed computing to tackle previously intractable problems in computational biology.* **Larson, S., y otros, y otros.** s.l. : Computational Genomics, 2003.

48. **A, Beck.** *High Throughput Computing: An Iinerview With Miron Livny.* s.l. : HPCwire, 1997.

49. *Parallel Models for Particle Swarm Optimizers.* **Belal, M. y El-Ghazawi, Talbi.** 1, Washington : s.n., Enero de 2004, Vol. 4.

50. **Pit, Laurens Jan.** *Parallel genetic algorithms.* s.l. : Leiden University, 1995. Tesis de maestría.

51. *A massively parallel genetic algorithm.* **Spiessens, P y Manderick, B.** 1991. págs. 279–286. Proceedings of the 4th International Conference on Genetic Algorithms.

52. *MALLBA: A library of skeletons for combinatorial optimisation.* **Alba, Enrique, y otros, y otros.** 2002.

53. [En línea] [Citado el: 2 de Octubre de 2011.] http://www.cilib.net/.

54. [En línea] [Citado el: 2 de Octubre de 2011.] http://paradiseo.gforge.inria.fr.

55. **Clerc, Maurice.** *Standard Particle Swarm Optimisation.* 2011.

56. **Becerril C., Francisco.** *Java a su alcance.* s.l. : McGraw-Hill Interamericana, 1998.

57. Netbeans. [En línea] 20 de 10 de 2011. http://www.netbeans.org/.

58. Unified Modeling Language. [En línea] [Citado el: 27 de Octubre de 2011.] http://www.uml.org.

59. [En línea] [Citado el: 1 de 12 de 2011.] http://www.cyta.com.ar/biblioteca/bddoc/bdlibros/proyectoinformatico/libro/c5/c5.htm.

60. Increase productivity, comunication, and collaboration using UML visual modeling plataform. [En línea] [Citado el: 10 de Diciembre de 2011.] http://www.visual-paradigm.com/product/vpuml.

61. Springsource.org. [En línea] [Citado el: 5 de Mayo de 2012.] http://www.springsource.org/.

62. **Talbi, El-Ghazali.** *Metaheuristics from design to implementation.* New Jersey : Addison Welsley&Sons,Inc.,Publication, 2009.

63. Gonzalo Álvarez Marañón . *Java.* [En línea] CSIC. Todos los derechos reservados, 1997-1999 . [Citado el: 10 de 1 de 2011.] http://www.iec.csic.es/criptonomicon/java/quesjava.html.

64. *A Dissipative Particle Swarm Optimization.* **Wen-Jun , Zhang, Xiao-Feng, Xie y Zhi-Lian , Yang.** Hawaii : s.n., 2002. págs. 1456-1461. Congress in Evolutionary Computation.

65. *Data Clustering: A Review.* **Jain, Anil, Flynn, P J y Murty, M N.** 3, s.l. : ACM Computing Surveys, 1999, Vol. 31.

66. *Curve clustering with spatial constraints for analysis of spatio temporal data.* **Blekas , K., y otros, y otros.** Washington DC. : IEEE Computer Society, 2007. págs. 529-535.

67. **Mahajana, Meena, Nimbhorkara, Prajakta y Varadarajanb, Kasturi.** *The Planar k-means Problem is NP-hard.* Chennai : The Institute of Mathematical Sciences, 2009.

68. **Villagra, Andrea.** *Metaheurísticas aplicadas a Clustering.* San Luis : Departamento de Informática,Ejército de los Andes 950.

69. *Linearly Decreasing Weight Particle Swarm with Accelerated Strategy for Data Clustering.* **Yang, Cheng-Hong, Hsiao, Chih-Jen y Chuang, Li-Yeh.** s.l. : IAENG International Journal of Computer Science, 2010.

70. DICCIONARIO DE LA LENGUA ESPAÑOLA - Vigésima segunda edición. [En línea] [Citado el: 18 de Noviembre de 2011.] http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=taxonom%C3%ADa.

71. **Engelmore, R.S. y Morgan, A.** *Blackboard Systems.* s.l. : Addison-Wesley, 1988.

72. [En línea] [Citado el: 5 de Octubre de 2011.] http://www.cecill.info/index.en.html.

73. *Greedy randomized adaptive search procedures.* **Feo, T. A. y Resende, M. G. C.** 1995.

74. *Guided local search.* **Voudouris, C. y Tsang, E.** 1999.

75. **G Couloris , JD y Kinberg , T.** *Distributed Systems - Concepts and Design, 4th Edition.* 2001. Addison-Wesley, Pearson Education.

76. **R, Korf.** *Depth-first iterative-deepening: An optimal admissible tree search.* s.l. : Artificial Intelligence, 1985. págs. 97-109.

77. **Goldberg, D. E.** *Optimization and Machine Learning.* Massachussets : s.n., 1989.

78. **Marañón, Gonzalo Álvarez.** Gonzalo Álvarez Marañón. *Java.* [En línea] CSIC. Todos los derechos reservados, 1997-1999. [Citado el: 10 de 1 de 2011.] http://www.iec.csic.es/criptonomicon/java/quesjava.html.

79. **Arora, Sanjeev y Barak, Boaz .** *Computational Complexity: A Modern Approach.* 2007.

80. **Papadimitriou, C. H.** *The complexity of combinatorial optimization problems.* s.l. : Princeton University, 1976. Master's Thesis.

81. *The Performance of Bags-of-Tasks in Large-Scale Distributed Systems.* **Iosup, Alexandru, Sonmez, Ozan y Anoep, Shanny .**

82. *Particle Swarm Optimizationin High-Dimensional Bounded Search Spaces.* **Helwig, Sabine y Wanka, Rolf .** 2007. págs. 198-205. In Proceedings of the 2007 IEEE Swarm Intelligence Symposium.

83. *A correction. On computable numbers, with an application to the entscheidungs problem.* **Turing, Alan.** 1938. Proceedings of the London Mathematical Society.

84. **Poli, Riccardo.** *An Analysis of Publications on Particle Swarm Optimisation Applications.* 2007. Reporte técnico.

85. **Keane, Thomas.** *A General-Purpose Heterogeneous Distributed Computing System.* s.l. : National University of Ireland Maynooth, 2004. Tesis de maestría.

86. *Particle swarm optimization: developments,applications and resources.* **Eberhart, Russell y Shi, Yuhui.** 2001. Vol. 1, págs. 81-86.

87. *Comparing inertia weights and constriction factors in particle swarm optimization.* **Eberhart, Russell y Shi, Yuhui.** 2000, Vol. 1.

88. **Sudkamp, Thomas.** *Languages and Machines: An introduction to the Theory of Computer Science.* s.l. : Addison Wesley, 2005.

89. *An Empirical Comparison of Parallel and Distributed Particle Swarm Optimization Methods.* **Vanneschi, Leonardo , Codecasa, Daniele y Mauri, Giancarlo .** Milán : s.n., 2010.

90. *Principles of Docking: An Overviewof Search Algorithms and a Guide to Scoring Functions.* **Halpering, H y Wolfson, Nussinov.** 2002, PROTEIN: Structure, Function, and Genetics, págs. 409-443.

Anexos

Anexo 1: Código de la clase SPSO2007VelocityUpdateStrategy

```
public class SPSO2007VelocityUpdateStrategy extends SPSO2006VelocityUpdateStrategy {

public SPSO2007VelocityUpdateStrategy()
{
}
@Override
public void updateVelocity(StandardParticle particle)
{
        Vector<Double> velocity = particle.getVelocity();

        Vector<Double> position = particle.getPosition();

        Vector<Double> bestPosition = particle.getBestPosition();

        Vector<Double> nBestPosition = particle.getBestNeighbor().getBestPosition();

        boolean isNeighborhoodBest = particle.equals(particle.getBestNeighbor());

        for (int i = 0; i < velocity.size(); i++)
        {
                double cognitiveComponent = PSOMaths.getUniform()*
                cognitiveFactor.getParameter()* (bestPosition.get(i) - position.get(i));

                double socialComponent = isNeighborhoodBest ? 0.0 : PSOMaths.getUniform() *
                socialFactor.getParameter() * (nBestPosition.get(i) - position.get(i));

                double velocityValue = inertia.getParameter() * velocity.get(i) + cognitiveComponent +
                socialComponent;

                velocity.set(i, velocityValue);
        }
}
}
```

Anexo 2: Estructura del fichero XML de configuración de la biblioteca.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"
xmlns:aop="http://www.springframework.org/schema/aop">

<bean id="configuration" class="pso.problem.PSOConfiguration">
<property name="dimention" value="10"/>
<property name="swarmSize" value="20"/>
<property name="topology" ref="topology"/>
<property name="inertia" ref="inertia"/>
<property name="velocityMax" ref="velocityMax"/>
<property name="positionMin" ref="positionMin"/>
<property name="positionMax" ref="positionMax"/>
<property name="cognitiveFactor" ref="cognitiveFactor"/>
<property name="socialFactor" ref="socialFactor"/>
<property name="initializationStrategy" ref="initializationStrategy"/>
<property name="positionUpdateStrategy" ref="positionUpdateStrategy"/>
<property name="velocityUpdateStrategy" ref="velocityupdatestrategy"/>
<property name="confinementStrategy" ref="confinementStrategy"/>
<property name="iterationStrategy" ref="iterationStrategy"/>
<property name="stoppingConditions">
<list>
<ref bean="maxiterations"/>
</list>
</property>
</bean>
<bean id="topology" class="pso.topologies.GBestTopology">
</bean>
<bean id="inertia" class="pso.parameters.PSOLinearDecreasingParameter">
<property name="lowerBound" value="0.4"/>
```

```
<property name="upperBound" value="0.9"/>
</bean>
<bean id="velocityMax" class="pso.parameters.PSOConstantParameter">
<property name="parameter" value="100"/>
</bean>
<bean id="positionMin" class="pso.parameters.PSOConstantParameter">
<property name="parameter" value="-100"/>
</bean>
<bean id="positionMax" class="pso.parameters.PSOConstantParameter">
<property name="parameter" value="100"/>
</bean>
<bean id="cognitiveFactor" class="pso.parameters.PSOConstantParameter">
<property name="parameter" value="2.05"/>
</bean>
<bean id="socialFactor" class="pso.parameters.PSOConstantParameter">
<property name="parameter" value="2.05"/>
</bean>
<bean id="initializationStrategy"
class="pso.initializationstrategies.PSORandomBoundedInitializationStrategy">
<property name="velocityMax" ref="velocityMax"/>
<property name="positionMax" ref="positionMax"/>
<property name="positionMin" ref="positionMin"/>
</bean>
<bean id="iterationStrategy" class="pso.algorithm.iterationstrategies.SynchronousIterationStrategy">
</bean>
<bean id="positionUpdateStrategy"
class="pso.particle.positionupdatestrategy.PSOStandardPositionStrategy">
</bean>
<bean id="velocityupdatestrategy"
 class="pso.particle.velocityupdatestrategy.PSOStandardVelocityStrategy">
 <property name="inertia" ref="inertia"/>
<property name="socialFactor" ref="socialFactor"/>
<property name="cognitiveFactor" ref="cognitiveFactor"/>
```

```
</bean>
<bean id="maxiterations" class="pso.algorithm.stoppingconditions.MaximumIterations">
<property name="maximumIterations" value="100"/>
</bean>
<bean id="confinementStrategy"
class="pso.algorithm.confinementstrategy.ClampingConfinementStrategy">
</bean>
</beans>
```

Glosario

PSO: Optimización por Enjambre de Partículas o Particle Swarm Optimization.

SA: Enfriamiento Simulado o Simulated Annealing.

TS: Búsqueda Tabú o Tabu Search.

VNS: Búsqueda con Vecindario Variable o Variable Neighborhood Search.

ACO: Optimización por Colonia de Hormigas o Ant Colony Optimization.

SS: Búsqueda Dispersa o Scatter Search.

GRASP: Procedimiento de Búsqueda Miope Aleatorizado y Adaptativo o Greedy Randomized Adaptative Procedure.

CASE: Ingeniería de Software Asistida por Computadoras.

UML: Lenguaje Unificado de Modelado.

J2EE: Java 2 Edición Empresarial o Java 2 Enterprise Edition.

LAN: Red de Área Local o Local Area Network.

WAN: Red de Área Amplia o Wide Area Network.

IDE: Entorno de Desarrollo Integrado.

GOMAD: Ordenamiento de Genes en Microarray de Datos o Gene Ordering Microarray Data.

RAM: Memoria de Acceso Aleatorio o Random Access Memory.