Universidad de las Ciencias Informáticas

Facultad 6

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

Título: Biblioteca para la Optimización basada en Colonias de Hormigas sobre la Plataforma T-arenal.

Autor: Reynaldo Ubieta Ramos

Tutores: MSc. Dannier Trinchet Almaguer

Ing. César Raúl García Jacas

La Habana, 29 de junio del 2012

"Año 54 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos 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 _____.

Reynaldo Ubieta Ramos

_____

Firma del Autor

MSc. Dannier Trinchet Almaguer

_____

Firma del Tutor

Ing. César Raúl García Jacas

_____

Firma del Tutor

DATOS DE CONTACTO

Autor:

Reynaldo Ubieta Ramos

Universidad de las Ciencias Informáticas, Habana, Cuba.

Email: rubieta@estudiantes.uci.cu

Tutor:

MSc. Dannier Trinchet Almaguer

Universidad de las Ciencias Informáticas, Habana, Cuba.

Email: trinchet@uci.cu

Tutor:

Ing. César Raúl Garcías Jacas

Universidad de las Ciencias Informáticas, Habana, Cuba.

Email: crjacas@uci.cu

AGRADECIMIENTOS

Primeramente quisiera agradecer a mis padres por brindarme siempre un apoyo incondicional, por la formación que me han proporcionado, por ser los principales pilares en mi vida. A mi hermana Lisset por su comprensión, dedicación y cariño brindado a lo largo de todos estos años. A mi familia por toda la ayuda brindada a lo largo de mi vida. A mi novia Anet por tolerarme y darme aliento en todo momento.

Gracias a mis tutores Trinchet y César por la colaboración brindada.

Desearía agradecer a una persona que a lo largo de toda mi carrera me han servido de apoyo y fuente de inspiración como Yasnelys Ochoa. A María Teresa y a Yisel Pompa por la ayuda brindada en los momentos donde más me hacía falta una mano amiga. A mis compañeros Contreras, Héctor, Pinal y Carlos por aguantarme, ayudarme y tolerarme en todo momento.

Mucha gracias a todos los profesores que a lo largo de mi carrera han contribuido a mi formación.

Agradecimientos para todas las personas que de una forma u otra han colaborado en mi formación como profesional y como ser humano.

DEDICATORIA

A mis padres por estar presentes en todo momento y por todo el amor brindado.

A mi hermana Lisset por ser alguien especial en mi vida.

Y en especial a mi abuela Manuela Puñales por ser siempre mi principal fuente de aliento, esperanza y fuerza en todos los tiempos.

RESUMEN

En la actualidad el hombre se enfrenta a diversos y difíciles problemas de optimización, los cuales requieren un amplio poder de cómputo y con su solución se lograría un mayor desarrollo en diversas esferas de la vida. Resolver estos problemas en una sola unidad de procesamiento resulta muy complicado, una de las alternativas más eficientes de hacerlo es mediante sistemas distribuidos. Para brindarle solución a estos problemas se han creado técnicas, entre las cuales se encuentran las técnicas metaheurísticas. Un ejemplo de ellas es la Optimización basada en Colonias de Hormigas.

La Universidad de las Ciencias Informáticas es un centro cubano de estudios universitarios que cuenta con varios proyectos productivos, entre ellos la plataforma de cálculos distribuidos T-arenal, la cual aprovecha la gran capacidad de cómputo con que se cuenta en ese centro, aprovechada para solucionar problemas de forma distribuida. En el presente trabajo se desarrolló una biblioteca en Java que cuenta con tres algoritmos de Optimización basados en Colonias de Hormigas que permiten resolver problemas de optimización de manera eficiente sobre la plataforma T-arenal. Para validar esta solución se realizaron pruebas a la biblioteca utilizando una instancia de Problema de Asignación Cuadrática donde se obtuvo resultados satisfactorios.

Palabras clave: biblioteca, eficiente, Optimización basada en Colonias de Hormigas, Problema de Asignación Cuadrática, problemas de optimización.

ÍNDICE

ÍNDICE

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El desarrollo de la humanidad se ha visto impulsado en gran medida por la necesidad de superar problemas en la realización de una tarea u objetivo. A medida que se fueron alcanzando estas metas, surgieron obstáculos más complejos, por lo que se hizo necesaria la búsqueda de nuevas y mejores vías de solución. Un ejemplo son los problemas de optimización. Estos se basan en la obtención de la mejor solución entre un conjunto de soluciones candidatas, ya sea minimizando o maximizando el valor de uno o varios parámetros en una actividad determinada.

Los métodos de optimización se pueden encontrar aplicados en diversas ramas como el transporte, la Bioinformática, las finanzas, entre otras. Estos se encuentran divididos en dos grandes grupos: los exactos y los aproximados. Pertenecientes al último grupo se encuentran las técnicas metaheurísticas, las cuales hoy día son muy utilizadas por ser eficientes y eficaces en la solución de grandes y complejos problemas de optimización.

Los algoritmos de Optimización basados en Colonias de Hormigas o *Ant Colony Optimization* (ACO por sus siglas en inglés) constituyen una novedosa técnica metaheurística, creada por Marco Dorigo a principios de la década de los 90. Su funcionamiento se basa en la observación del comportamiento de las hormigas, animales casi ciegos, pero con la habilidad de optimizar el camino hasta llegar a la fuente de su alimento y regresar al nido (1; 2).

Aunque con la implementación de técnicas metaheurísticas como ACO se resuelven problemas de optimización de gran tamaño y con gran calidad, en reiteradas oportunidades una sola unidad de procesamiento no es suficiente, debido al alto costo computacional que ellos requieren. Las supercomputadoras[1] son una solución acertada para darle respuesta a la necesidad de cómputo que requieren un gran número de los problemas anteriormente mencionados, pero tienen como dificultad un elevado costo en el mercado, por lo cual muchas instituciones no pueden poseerlas.

Una alternativa menos costosa es la utilización de sistemas distribuidos, esto consiste en el empleo de un conjunto de máquinas conectadas a una red compartiendo sus recursos para procesar información. De esta manera el tiempo de obtención de los resultados es mucho menor, comparado con la ejecución de los mismos cálculos en una sola unidad de procesamiento.

[1] Tipo de computadoras más potentes y más rápidas que existen en un momento dado. Son de gran tamaño y pueden procesar enormes cantidades de información en poco tiempo pudiendo ejecutar millones de instrucciones por segundo, están destinadas a una tarea específica y poseen una capacidad de almacenamiento muy grande.

La Universidad de las Ciencias Informáticas (UCI) constituye un centro cubano de altos estudios con gran magnitud en cuanto a infraestructura tecnológica y productiva, ya que es el centro a nivel nacional con el mayor número de computadoras personales, con alrededor de 6000 de ellas distribuidas en toda la red universitaria (3). La UCI tiene entre otros, el Centro de Tecnologías de Gestión de Datos (DATEC), el cual cuenta con un Departamento de Bioinformática, con varios años de experiencia en esa ciencia. Esta línea de investigación posee el proyecto Plataforma de Servicios de Bioinformáticos, que entre sus módulos tiene uno que se encarga de manejar el procesamiento de grandes volúmenes de información de forma distribuida, disminuyendo de esta manera el tiempo de procesamiento de los datos, dicho módulo es denominado Plataforma de Tareas Distribuidas (T-arenal). T-arenal aprovecha los recursos de cómputo (las PCs) que se encuentran disponibles en la UCI, siendo esto posible a través de una red local con una velocidad de transferencia de 100 Mbps, para solucionar problemas que requieran un elevado costo computacional.

Hoy en día T-arenal está siendo utilizada para solucionar problemas pertinentes a la Bioinformática, como el expuesto en el trabajo “Sistema de cómputo distribuido aplicado a la Bioinformática” (3), donde se brinda una solución a la búsqueda de posibles inhibidores para una proteína, por métodos conocidos como “Tamizaje Virtual”. También ha sido utilizada para resolver problemas de otra índole como se presenta en el trabajo “Algoritmos paralelos para el modelado de yacimientos lateríticos” (4).

T-arenal está construida de forma genérica, por lo que puede procesar diversos problemas que requieran un gran poder computacional, incluyendo los de optimización. No obstante, esta no cuenta con las herramientas necesarias que faciliten darle una solución de manera eficiente a grandes y complejos problemas de optimización, ya que resulta muy engorroso la modelación y resolución de estos, debido a que se debe realizar la implementación del algoritmo que se empleará para solucionarlos.

Por todo lo anteriormente planteado surge como **problema a resolver** de la presente investigación: ¿Cómo resolver problemas costosos computacionalmente utilizando algoritmos de Optimización basados en Colonias de Hormigas haciendo uso de los recursos de la Universidad de las Ciencias Informáticas?, el cual se centra en el **objeto de estudio**: algoritmos de Optimización basados en Colonias de Hormigas en entornos distribuidos, enmarcado en el **campo de acción**: algoritmo de optimización basado en Colonia de Hormigas sobre la plataforma de cálculo distribuido T-arenal.

Para solucionar el problema anteriormente planteado, se define el siguiente **objetivo general**: Implementar una biblioteca para resolver problemas costosos computacionalmente utilizando algoritmos de optimización basados en Colonias de Hormigas sobre la plataforma T-arenal.

La **hipótesis investigativa** para el presente trabajo es:

Si se emplean algoritmos de Optimización basados en Colonias de Hormigas sobre la plataforma de Tareas Distribuidas T-Arenal para resolver problemas de optimización, costosos computacionalmente, aprovechando la capacidad de cómputo de la UCI, es posible disminuir considerablemente el tiempo de ejecución de los mismos.

Y se determina como **variable de la investigación**:

- Tiempo de ejecución

Para darle cumplimiento al objetivo se trazaron una serie de **tareas de la investigación**:

1. Revisión bibliográfica del algoritmo de Optimización basados Colonias de Hormigas
2. Revisión de los esquemas distribuidos del algoritmo Optimización basados Colonias de Hormigas.
3. Estudio de los esquemas distribuidos del algoritmo Optimización basados Colonias de Hormigas.
4. Diseño de la biblioteca para el algoritmo de Optimización por Colonias de Hormigas.
5. Implementación de la versión secuencial de la biblioteca.
6. Implementación de la versión distribuida de la biblioteca.
7. Modelado y resolución de un Problema de Asignación Cuadrática (QAP, por sus siglas en inglés, *Quadratic Assignment Problem*).
8. Realización de pruebas a la aplicación de la biblioteca en la plataforma de tareas distribuidas T-arenal.

La **estructura del documento** se compone por tres capítulos:

Capítulo 1: Fundamento teórico. En este capítulo se introducen algunos conceptos de optimización, así como se muestran los métodos de optimización con sus clasificaciones y ejemplos de los mismos. Se profundiza en la Metaheurística de Optimización basada en Colonias de Hormigas. Además se abordan conceptos de computación distribuida y algoritmos de optimización por Colonias de Hormigas en entornos distribuidos.

Capítulo 2: Diseño e implementación de la Biblioteca ACO. Este capítulo comprende la realización del diseño de las clases utilizadas en la confección de la biblioteca sobre la plataforma T-arenal. Se

muestra la implementación de la propuesta de solución, incluyéndose algunas muestras del código implementado.

Capítulo 3: Resultados y discusión. En este capítulo se presentan los resultados obtenidos por la biblioteca, de forma secuencial y de forma distribuida, estos últimos ejecutados en la plataforma T-arenal, aplicados a un QAP, realizándose además el análisis de los mismos.

CAPÍTULO 1: FUNDAMENTO TEÓRICO

En este capítulo se introducen conceptos de optimización, se muestran los métodos de optimización con sus clasificaciones y ejemplos de los mismos. Se profundiza en la metaheurística de Optimización basada en Colonias de Hormigas. Además de abordar conceptos de computación distribuida y algoritmos de Optimización por Colonias de Hormigas en entornos distribuidos.

1.1 Problemas de optimización

Los problemas de optimización se corresponden con la tupla (S, f) siendo S el espacio de búsqueda, f la función objetivo que asigna valores $f(s)$ a $s \in S$ y Los problemas de optimización pueden ser de minimización o maximización de la función objetivo (5; 6). Existen problemas de optimización que poseen más de una función objetivo, los cuales se conocen como problemas de optimización multiobjetivo.

Los problemas se dividen en dos categorías de acuerdo a las variables de decisión asociadas a las soluciones, estas pueden tomar valores reales o discretos. Cuando los valores son reales resultan los problemas de optimización de dominio continuo. Cuando los valores son discretos resultan los problemas de optimización de dominio discreto. Los problemas de optimización combinatoria están comprendidos dentro de los de dominio discreto y corresponden a encontrar un objeto dentro de un conjunto finito o infinitamente contable (7; 8). Este objeto puede ser un número entero, un subconjunto de ellos, una permutación, o una estructura de grafo. Un problema de optimización combinatoria sería $P = (S, f, \Omega)$ y puede ser definido por (9):

- X_i es un conjunto de variables discretas con valores $x_i \in D_i = \{d_i, \ldots, d_{|D_i|}\}$,

  $i = 1, \ldots, n$;
- Ω es un conjunto de restricciones entre las variables;
- La función objetivo del problema a minimizar es $f: D_i \times \ldots \times D_n \to \mathbb{R}$.

El espacio de búsqueda S_Ω es definido por:

$$S_\Omega = \{s = \{(X_1, x_1), .., (X_n, x_n)\} | x_i \in D_i, s\ satisface\ \Omega\}$$

Y puede verse como el conjunto de posibles soluciones candidatas del problema a solucionar. Resolver un problema de optimización combinatoria consiste en encontrar una solución factible $s^* \in S$, Siendo s^* una solución óptima global del problema P, tal que:

$$f(s^*) \leq f(s), \forall x \in S_\Omega.$$

1.2 Métodos de optimización

Los problemas de optimización combinatoria están presentes en diversos campos como la Economía, el Comercio, la Ingeniería, la Industria o la Medicina. Sin embargo, a menudo estos problemas son muy difíciles de resolver en la práctica. En el presente, siguen apareciendo nuevos problemas de este tipo, lo que ha dado lugar a muchas propuestas de algoritmos para tratar de solucionarlos. Estos algoritmos son clasificados en dos grupos: exactos y aproximados. (10; 11).

Los métodos exactos garantizan obtener la solución óptima y tratan de garantizar que esta solución sea la óptima global. El inconveniente de estos métodos es que el tiempo necesario para la resolución de los problemas, crece exponencialmente con el tamaño de los mismos. Entre los algoritmos de este tipo se encuentran: proceso de vuelta a atrás (*back-tracking*), Ramificación y Acotación (*Branch and Bound*), programación dinámica, entre otros.

Los métodos aproximados (o heurísticos) tienen como fundamento sacrificar la garantía de encontrar la solución óptima para el bien de obtener buenas soluciones en una reducción de tiempo significativa. Ellos se clasifican principalmente en heurísticos constructivos y algoritmos de búsqueda local.

1.2.1 Heurísticos constructivos

Los heurísticos constructivos, son el típico método de rápida aproximación. Ellos generan las soluciones partiendo desde una vacía, a la cual se le añaden oportunamente los componentes, hasta generar una solución completa o se satisface un criterio de parada. Las heurísticas constructivas tienen como inconveniente que la mayor parte de las ocasiones retornan soluciones de inferior calidad debido a la gran dependencia que tienen con el problema, y para su descripción se debe tener gran dominio del mismo. Ocurre también que en problemas que poseen muchas restricciones puede que las soluciones parciales generadas, conduzcan a soluciones no factibles (5).

1.2.2 Algoritmos de búsqueda local

Los algoritmos de búsqueda local comienzan desde una solución inicial y tratan iterativamente de remplazar la solución actual por una mejor perteneciente a una vecindad, hasta encontrar un óptimo local. La declaración de una buena vecindad garantiza un buen porciento de la buena solución del problema. La vecindad de una solución $s \in S$ (siendo S el espacio de búsqueda), denotada por $N(s) \subset S$, es todo el conjunto de soluciones que se pueden construir a partir de s, aplicando un operador específico de modificación, conocido como movimiento. En función del operador de

movimiento utilizado, el vecindario cambia y el modo de explorar el espacio de búsqueda también, pudiendo la búsqueda complicarse o simplificarse. Las soluciones que son óptimos locales pueden ser vistas como las “mejores” dentro de un espacio de búsqueda acotado (6).

1.3 Técnicas Metaheurísticas

Las metaheurísticas surgen en los años 1970, como algoritmos basados en métodos heurísticos, con un alto nivel de aprovechamiento y eficiencia en la exploración de los espacios de búsquedas en los problemas. La palabra metaheurística proviene del idioma griego y es la composición de dos palabras: “Heurística” que proviene del verbo “*heuriskein”*, que significa “encontrar” y el sufijo “meta” que significa “nivel superior” (12; 5; 8).

Las metaheurísticas se han difundido actualmente por su gran capacidad para solucionar problemas complejos. Estas técnicas han demostrado que son eficientes y eficaces al enfrentarse a problemas grandes, garantizando soluciones satisfactorias en un tiempo razonable. Debido a estas razones se utilizan en muchas esferas:

- Para la construcción de máquinas inteligentes; minería de datos (*data mining*); en la Bioinformática y en las finanzas.
- La creación de sistemas para el modelado, simulación e identificación en áreas como la física, la química y la biología.
- La creación de programas para áreas de la ciencia como la robótica y en sectores importantes en la vida del hombre como la trasportación y la medicina.

La construcción de técnicas metaheurísticas se basa en dos criterios contradictorios: la exploración del espacio de búsqueda (diversificación) y la explotación de las mejores soluciones (intensificación). En la diversificación se busca encontrar zonas no exploradas, para así ampliar el rango de búsqueda de nuevas regiones y no restringir el área de búsqueda en el problema. Mientras con la intensificación se garantiza explorar a fondo las soluciones más prometedoras (5).

Los algoritmos metaheurísticos por sus características son clasificadas en dos grupos: algoritmos basados en trayectoria y los algoritmos basados en poblaciones (ver Figura 1).

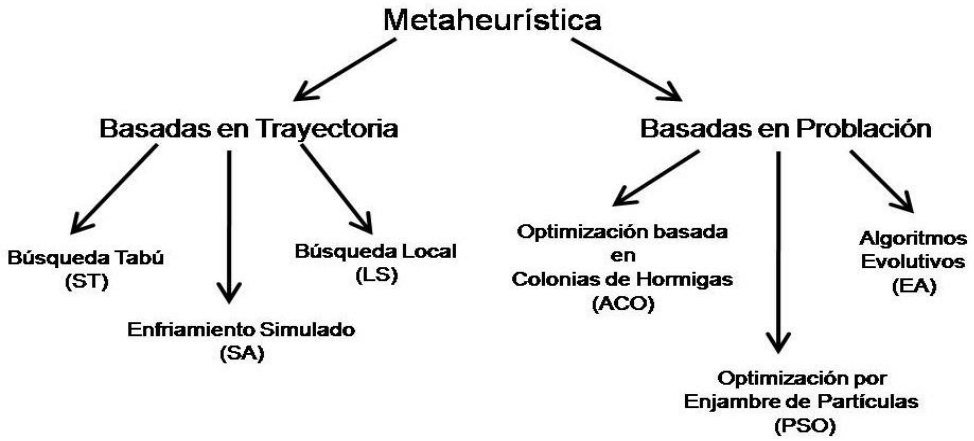


Figura 1: Clasificación de las metaheurísticas.

1.3.1 Metaheurísticas basadas en trayectoria

Estas técnicas se basan en la mejora de la solución obtenida en la resolución de un problema. La principal característica de estos métodos es que parten de un punto y van renovando la solución actual mediante la exploración de la vecindad, confeccionando una trayectoria a través del espacio de búsqueda generado (ver Figura 1). 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 mínimos locales. Esto implica la necesidad de una condición de parada más compleja que la de encontrar un mínimo local. Alcanzar un número máximo de iteraciones predefinidas, encontrar una solución con una calidad aceptable y consumir un tiempo máximo de CPU son algunos de los criterios de paradas más utilizados (8).

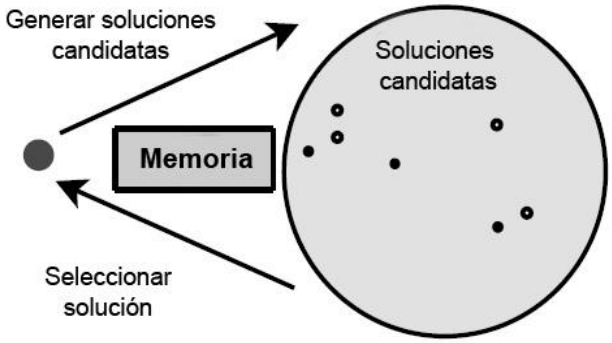


Figura 2: Funcionamiento básico de los métodos metaheurísticos basados en trayectoria.

Se tienen los siguientes algoritmos con dichas características:

- **Enfriamiento Simulado o *Simulated Annealing* (SA):** aplicado a problemas de optimización surgidos del trabajo de S. Kirkpatrick (13) y V Cerny (14). Es uno de los más antiguos entre las metaheurísticas y seguramente es el primer algoritmo con una estrategia explícita para escapar de los mínimos locales. La idea del SA es simular el proceso de recocido del metal y del cristal. Para evitar quedar atrapado en un mínimo local, el algoritmo permite elegir una solución cuyo valor de la función objetivo sea peor que el de la solución actual (15).
- **Búsqueda Local o *Local Search* (LS):** Es una de las metaheurísticas más antiguas y sencillas (16). Se caracteriza por buscar una solución inicial en una vecindad. En cada iteración que se realiza, la heurística sustituye a la actual solución por un vecino que genera una mejor solución de la función objetivo. Esta técnica se detiene cuando los vecinos candidatos son peores que la solución seleccionada, con esto se logra obtener un óptimo local (6)
- **Búsqueda Tabú o *Search Tabu* (TS):** TS es una de las metaheurísticas más exitosa en la aplicación de problemas de optimización combinatoria. La idea de este algoritmo es la utilización explícita de una historia de búsqueda para escapar de mínimos locales y para llevar a cabo una estrategia explorativa. Un algoritmo TS utiliza una memoria a corto plazo para escapar de los mínimos locales y evitar los ciclos. El término de memoria corta se lleva a cabo a través de una lista tabú (TL), que es la encargada de guardar las soluciones recientemente visitadas y las excluye de la vecindad de la solución actual. En cada iteración la mejor solución es seleccionada como óptima loca y es agregada a la lista tabú (17; 18).

1.3.2 Metaheurísticas basadas en población

Estas metaheurísticas se pueden ver como una mejora iterativa de una población de soluciones. Para comenzar, las poblaciones son inicializadas y cada nueva generación de poblaciones genera nuevas soluciones. En este proceso iterativo de generación de una nueva población, se realiza el reemplazo de la población actual por la nueva generada a través de un procedimiento de selección. Este proceso iterativo se realiza hasta que una condición de parada se cumpla. En todas las fases de generación y reemplazo de poblaciones es utilizada una memoria como se refleja en la Figura 3. Muchas de las metaheurísticas basadas en poblaciones son bioinspiradas[2] (6).

[2] Basadas en procesos que ocurren en la naturaleza.

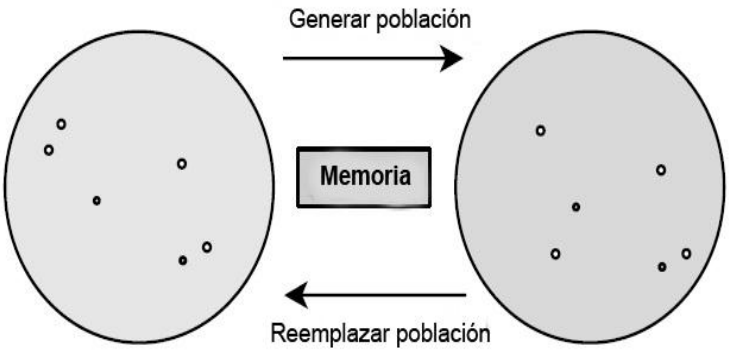


Figura 3: Funcionamiento básico de los métodos metaheurísticos basados en población.

Algunos de estos algoritmos son:

- **Algoritmos Evolutivos o *Evolutionary Algorithms* (EA):** están inspirados en los principios de la genética y la selección natural. Cada individuo representa una solución potencial al problema que se está resolviendo. Estos algoritmos hacen evolucionar una población de individuos sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución donde se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados. La modificación de la población se lleva a cabo mediante tres operadores: selección, cruzamiento y mutación (12; 19).
- **Optimización por Enjambre de Partículas o *Particle Swarm Optimation* (PSO):** técnica inspirada en el comportamiento social del vuelo de las bandadas de aves o el movimiento de los bancos de peces. Se fundamenta en los factores que influyen en la toma de decisión de un agente que forma parte de un conjunto de agentes similares. La toma de decisión por parte de cada agente se realiza conforme a una componente social y una componente individual, mediante las que se determina el movimiento (dirección) de este agente para alcanzar una nueva posición en el espacio de soluciones. Simulando este modelo de comportamiento se obtiene un método para resolver problemas de optimización (20; 21)
- **Optimización basada en Colonias de Hormigas o *Ant Colony Optimization* (ACO):** esta metaheurística es detallada a partir del epígrafe 1.4. 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 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 (22). El segundo elemento tiene que ver con la efectividad del método en la resolución de problemas de optimización de

naturaleza discreta, 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 Optimización por Enjambre de Partículas respectivamente.

1.4 Metaheurística de Optimización basada en Colonias de Hormigas

Las colonias de hormigas, y en general las sociedades de insectos sociales, son sistemas distribuidos que, a pesar de la sencillez de sus individuos, presentan una organización social altamente estructurada. Los estudios derivados de la observación del comportamiento de las hormigas reales se han utilizado como una fuente de inspiración para el diseño de algoritmos metaheurísticos para la solución de problemas de optimización.

1.4.1. Las Colonias de Hormigas Naturales

Las hormigas son insectos sociales que viven en colonias. Mediante la colaboración mutua que existe entre ellas, demuestran complejos comportamientos y realizan tareas difíciles desde el punto de vista de las capacidades de una hormiga individual. En su comportamiento, resalta la habilidad que poseen ellas para encontrar en su andar entre el hormiguero y la fuente de alimentación la vía más corta. Resulta verdaderamente interesante el hecho de que muchas especies de hormigas son casi ciegas, por lo cual no pueden utilizar pistas visuales (1).

Cuando una hormiga se desplaza entre el hormiguero y una fuente de alimentación, desprenden una sustancia química denominada feromona. Esta sustancia influyen en la decisión de una hormiga en el momento de elegir cuál camino tomar; los caminos que mayor concentración de feromona contengan, tienen más probabilidades de ser elegidos por una hormiga. En el caso que no encuentre ningún rastro de feromona, la hormiga se moverá básicamente de manera aleatoria. Experimentos realizados por biólogos, han demostrado que las hormigas prefieren de manera probabilística los caminos marcados con una concentración superior de feromona (1). Esta forma de coordinación de sus actividades es conocida como estimergia (*stigmergy)* (23; 24), la cual es una vía de comunicación indirecta, basada en la modificación del medio ambiente por un individuo y la aceptación al cambio por los demás agentes.

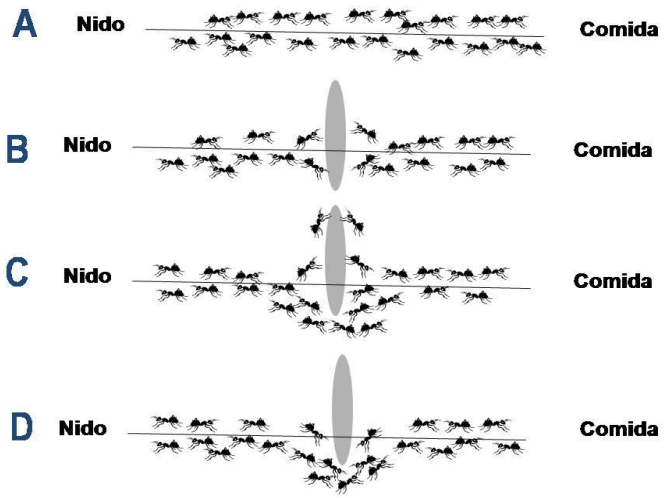


Figura 4: Recorrido que realizan las hormigas naturales en busca del camino más óptimo del nido a la fuente de alimentación.

En la Figura 4 se evidencia cómo este mecanismo permite a las hormigas encontrar el camino más corto entre dos puntos. Al comienzo existen dos caminos por los cuales las hormigas van transitando (sección A, Figura 4). Luego se les incorpora un obstáculo y de esa forma las hormigas comienzan a elegir su ruta de forma probabilística (sección B y C, Figura 4). El camino que se forma por debajo del obstáculo es más corto que el que se forma por encima, por lo tanto el tiempo en su recorrido será menor y tendrá una mayor concentración de feromona debido a que será mayor la cantidad de hormigas que transiten por él en ambos sentidos (sección D, Figura 4). La evaporación del rastro de feromona será mayor en el camino por el que menos hormigas transiten.

1.4.2 Algoritmos de Optimización basados en Colonias de Hormigas

La idea de los algoritmos de Optimización basados en ACO es imitar el comportamiento de las hormigas reales para resolver problemas de optimización (25). Este puede ser visto como un sistema de muchos agentes imitando el comportamiento de una colonia de hormigas, en la cual cada agente simula la manera de comportarse de una hormiga real. Una colonia de hormigas artificiales, trabajan de manera cooperativa y se comunican mediante rastros de feromona artificiales. Los algoritmos de ACO son básicamente algoritmos constructivos utilizados para solucionar problemas de optimización combinatoria.

Hormigas Artificiales

La hormiga artificial es un agente computacional simple que intenta construir posibles soluciones a un problema mediante el recorrido de un grafo llamado grafo de construcción $G = (N, A)$, utilizando

la explotación de los rastros de feromona disponibles y la información heurística disponibles en el grafo. El grafo de construcción G (totalmente conectado) consiste en un conjunto de componentes N, representados por los nodos del grafo G y un conjunto de arcos A. No en todos los casos las soluciones construidas son válidas, para comprobar los resultados obtenidos se crean criterios que verifican dichas soluciones y en caso de no ser correctas son penalizadas o descartadas. Una hormiga artificial tiene las siguientes características (1) (26):

- Tiene una memoria L que almacena información sobre el camino construido hasta el momento de manera secuencial.
- Tiene un estado inicial $\delta_{inicial}$ y se mueve hacia estados válidos confeccionando una solución. Este estado inicial puede seleccionarse de forma aleatoria o siguiendo algunos criterios en dependencia del problema a solucionar.
- Cuando se encuentra en un estado $\delta_r = < \delta_{r-1}, r >$, significa que proviene de un estado de la secuencia δ_{r-1} y se encuentra en el estado r pudiendo alcanzar cualquier estado perteneciente a la vecindad $N(r)$.
- La hormiga se mueve aplicando un criterio conocido como regla de transición, la cual es una función que valora los valores de feromona y/o los valores de la información heurística que se encuentran en el camino seleccionado y las restricciones del problema.
- El proceso de construcción de la solución concluye cuando se satisface una condición de parada del problema.
- La hormiga puede actualizar el camino de feromona en el transcurso de la construcción de la solución (actualización paso a paso) o cuando termine de generar una solución (actualización a posteriori).

1.4.3 Funcionamiento y estructura de ACO

El funcionamiento básico de un algoritmo ACO sigue las siguientes características generales:

Las hormigas m de la colonia artificial se mueven de manera probabilística mediante una regla de transición por los nodos adyacentes del grafo G, valorando la información local. Este movimiento generado por cada hormiga artificial sobre el grafo, va conformando una posible solución al problema, este proceso concluye cuando se cumple algún criterio de parada decido para el problema a resolver. Esta posible solución es valorada para confirmar que sea una solución válida al problema.

Mediante el proceso de “construcción de la solución” la hormiga puede actualizar el rastro de feromona o puede hacerlo luego de terminar todas sus transiciones por los estados, en

dependencia del criterio que se utilice. El algoritmo también incluye la evaporación de la feromona, evento mediante el cual los valores de feromonas son disminuidos con el propósito de lograr una ampliación del espacio de búsqueda y evitar un estancamiento en la búsqueda de una posible solución al problema. Los pasos de depósito y evaporación de los rastros de feromona son conocidos como "actualización de la feromona". El algoritmo también incluye un procedimiento que es conocido como la "acciones de búsqueda local", el cual es opcional, y es el encargado de acciones de manera global sobre el algoritmo ACO.

A continuación se representa un pseudocódigo genérico de la metaheurística ACO:

Procedimiento ACO

Repetir

Para cada hormiga **Hacer**

Construcción de la Solución

Fin Para

Acción de Búsqueda Local {Opcional}

Actualización de la Feromona

Hasta Criterio Parada

Fin Procedimiento

Los algoritmos ACO según lo anteriormente planteado tienen tres operaciones fundamentales las cuales son: construcción de la solución, actualización de la feromona y la acción de búsqueda local (opcional) (27).

Construcción de la solución en un algoritmo ACO

La construcción de una solución por una hormiga artificial está basada en la "regla de transición de estado". Estas hormigas pueden ser consideradas como procedimientos estocásticos (aleatorio) que construyen una solución de forma probabilística mediante la adición de componentes (nodos) a la solución, siempre que el movimiento entre los estados cumplan con la regla antes mencionada. La visita a los nodos se realiza de forma concurrente y asincrónica de los estados adyacentes que pertenecen a la vecindad en un grafo de decisión (o un grafo de construcción) G de un problema (1).

Para la construcción de un camino (o posible solución) son considerados los siguientes criterios:

- Información de la feromona (τ).
- Información heurística (η).

La información de la feromona es el componente fundamental de los algoritmos ACO. Esto consiste en generar un vector τ llamado camino (o rastro) de la feromona. El valor de la feromona $\tau_i \in \tau$ refleja información relevante en la construcción de una solución a un problema debido a que brinda las características de lo "buena" que han resultado otras soluciones generadas y sirve para generar nuevas soluciones. Esto consiste en que si una conexión entre dos nodos o un camino determinado contiene elevados valores de feromona significa que varias hormigas han pasado por él y esto generalmente converge en una buena solución. Esta información de la feromona se actualiza de forma dinámica mediante la búsqueda de una solución ya que representa la memoria del proceso de búsqueda de una hormiga (2).

La información heurística está en dependencia del problema a resolver. Esta información brinda más criterios de decisión a la hora de generar una solución por alguna hormiga. La incorporación de la información heurística se realiza con el objetivo de garantizar la diversificación de la búsqueda.

La construcción de una solución por una hormiga k generalmente comienza con la asignación de un nodo i de manera arbitraria. Una hormiga decide hacia cual nodo j trasladarse de acuerdo a la función probabilística,

$$p_{ij} = \frac{\tau_{ij}}{\sum_{k \in N} \tau_{ik}}, \forall j \in N \quad (1)$$

Donde p_{ij} es la probabilidad de una hormiga moverse desde el nodo *i* hasta el nodo j; N significa los nodos alcanzables (vecindad) por el nodo *i* que todavía no se han visitado por la solución en el grafo G; τ_{ij} representa la cantidad de feromonas que hay depositadas en el arco a_{ij}.

Existen problemas en los cuales la información heurística η_{ij}, que existe entre los nodos i y j es definida y es considerada para la construcción de la solución. La función probabilística que considera estos valores es la siguiente:

$$p_{ij} = \frac{{\tau_{ij}}^{\alpha} \times {\eta_{ij}}^{\beta}}{\sum_{k \in N} \tau_{ik} \times \eta_{ik}}, \quad \forall j \in N \quad (2)$$

Donde α y β son valores que reflejan la influencia de los valores de feromona e información heurística respectivamente, en el problema. Si $\alpha = 0$ los nodos con preferencia heurística tienen mayor probabilidad de ser escogidos, convirtiendo el algoritmo similar a un algoritmo voraz (*greedy*

en inglés) probabilístico clásico. Mientras si $\beta = 0$, solamente se tienen en cuenta los valores de feromona en el momento de realizar la construcción de la solución, esto provoca que todas las hormigas converjan en las mismas soluciones. Para evitar esta situación es necesario generar una adecuada proporcionalidad entre ambos valores (6).

Actualización de la feromona

Proceso mediante el cual los valores de feromonas son modificados. Estos valores pueden incrementarse, cuando una hormiga deposita esta sustancia en una arista por la que ha transitado, o disminuir mediante la evaporación de la feromona. Este proceso consta de dos partes:

- **Depósito de feromona:** se actualiza el valor de la feromona de acuerdo a la solución generada. Existen varias vías de realizar este proceso (28):
  - Actualización en línea de feromona paso a paso (*Online step-by-step pheromone update*, en idioma inglés): La hormiga va actualizando el rastro de feromona τ_{ij} en cada paso de la construcción de la solución.
  - Actualización en línea de retraso de feromona (*Online delayed pheromone update*, en idioma inglés): Se actualiza el rastro de feromona τ cuando una solución es generada completamente. A esta manera se le conoce como actualización a posteriori.
  - Actualización fuera de línea de feromona (*Off-line pheromone update*, en idioma inglés): El rastro de feromonas es actualizado cuando todas las hormigas generan una solución completa. Esta vía es la más utilizada por diferentes estrategias de ACO.
- **Evaporación de feromona:** el rastro de feromona es reducido, influenciado por un factor constante. El valor de este proceso está determinado por la fórmula:

$$\tau_{ij} = (1 - \rho)\tau_{ij} \quad (3)$$

Donde $\rho \in (0,1]$ es la tasa de evaporación de feromona. La evaporación tiene como principal objetivo que las hormigas no converjan en una solución rápidamente y lograr una diversificación del campo de búsqueda (29) (30) (31).

Las acciones de búsqueda local en los algoritmos ACO

Las acciones de búsqueda local son acciones opcionales y se realizan desde una óptica global, que no pueden ser ejecutadas por las hormigas debido al enfoque local que ellas ofrecen. Estas son tareas implementadas que no tienen un contrapunto natural.

Estas acciones ayudan al proceso de búsqueda de la solución más óptima del algoritmo. Ejemplos de estas acciones son:

- La activación de un procedimiento de optimización local.
- La observación de la calidad de todas las soluciones generadas y el depósito de feromonas extra en algunas transiciones de soluciones generadas.
- La búsqueda local de todas las soluciones generadas por las hormigas para seleccionar la más factible antes de realizar la actualización de la feromona.

Estas acciones pueden tomar un papel relevante en el algoritmo en el momento de buscar las solucione más factibles a un problema. (1)

Otros aspectos importantes en un algoritmo ACO

Los algoritmos ACO cuentan con un primer paso que es la "inicialización de los parámetros", ahí se inicializan de forma correcta los parámetros fundamentales de esta metaheurística. En este proceso se le otorga un valor inicial al rastro de feromona (τ_0) entre cada conexión. Este valor debe ser el mismo (generalmente) para todas las transiciones para así garantizar que todos los caminos tengan la misma "deseabilidad".

Otros aspectos a tener en cuenta son la tasa de evaporación (ρ), a la cual se le debe asignar un valor constante; la cantidad de hormigas artificiales (k) con que cuenta la colonia artificial; y los valores de α y β (1).

1.4.4 Diferentes algoritmos basados en la metaheurística ACO

Actualmente existen diversos algoritmos que siguen la metaheurística ACO. Entre los algoritmos ACO se encuentran: Sistema de Hormigas (*Ant Systems* o AS en inglés), Sistema de Colonias de Hormigas (*Ant Colony System* o ACS en inglés), Max-Min Sistema de Hormigas (*Max-Min Ant System* o MMAS en inglés) y el Sistema de la Mejor-Peor Hormiga (*Best-Worst Ant System* o BWAS en inglés) entre otros.

En este trabajo se abordarán tres de estos algoritmos: AS, ACS y MMAS. Estos se encuentran entre los principales algoritmos de la metaheurística ACO para el tratamiento de problemas de optimización combinatoria NP-duros.

Ant System

El algoritmo Sistema de Hormigas fue creado por Dorigo y Colirni en el año 1991 (32) (33). En un inicio se presentaron tres variantes diferentes: *AS-density*, *AS-quantity* y *AS-cycle*. La diferencia entre estas variantes recae en la forma de actualización de los rastros de feromona. En *AS-density* y *AS-quantity*, las hormigas realizan el depósito de feromona en la medida que construyen su solución (actualización en línea paso a paso). Mientras que en *AS-cycle* se realiza el depósito de

feromonas cuando todas las hormigas han generado una solución, y la proporción de feromonas depositadas está en relación con la calidad de la solución construida. Esta última variante es la que mejores resultados obtenía y en la literatura se le conoce como AS por este motivo.

La construcción de la solución para este algoritmo se realiza de la siguiente forma:

Inicialmente las hormigas (m) son ubicadas en nodos seleccionados aleatoriamente y de esta forma comienza a construir sus soluciones de manera concurrente. En cada paso de la construcción de la solución de cada hormiga k, se decide de forma probabilística a que nodo se trasladará, mediante la siguiente fórmula (26):

$$\rho_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}]^{\alpha} \times [\eta_{ij}]^{\beta}}{\sum_{f \in N_i^k} [\tau_{if}]^{\alpha} \times [\eta_{if}]^{\beta}}, & si\ j \in N_i^k \\ 0, & en\ otro\ caso \end{cases} \quad (4)$$

Donde η_{ij} es el valor asociado a la información heurística que existe entre los nodos i y j; N_i^k es el vecindario alcanzable por la hormiga k desde el nodo i, y N_i^k se refiere a todos los nodos que no ha sido visitado por ella. Una vecindad factible N_i^k se puede determinar a través de la memoria de la hormiga L^k, la cual consiste en la secuencia ordenada de nodos visitados por la hormiga k.

La actualización del rastro de feromona, para este algoritmo, se realiza después que todas las hormigas construyan una solución. Primero se evaporan los rastros de feromona pertenecientes a cada arista. La evaporación se realiza mediante un factor constante y se calcula de la siguiente manera:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} \quad (5)$$

Donde $\rho \in (0,1]$ es la tasa de evaporación. El parámetro ρ es utilizado para evitar una acumulación ilimitada de los rastros de feromonas y permite al algoritmo “olvidar” malas decisiones tomadas anteriormente. Al incrementarse el número de iteraciones, el valor de feromona de las aristas que no han sido elegidas por las hormigas, decrecen exponencialmente. Al concluir la evaporación, todas las hormigas depositan feromona en las aristas que cada una ha seleccionado en su solución:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \quad (6)$$

Donde $\Delta\tau_{ij}^{k}$ representa la cantidad de feromona que la hormiga k deposita en las aristas que ha visitado. Y se define como:

$$\Delta\tau_{ij}^{k} = \begin{cases} {}^{1}/_{C^k}, & si\ la\ arista(i,j)\ \in L^k \\ 0, & en\ otro\ caso \end{cases} \quad (7)$$

Donde C^k representa el costo de la solución generada (L^k). Con la operación anterior se garantiza que mientras mejor sea la solución generada por la hormiga, mayor será la cantidad de feromona depositada en las aristas del camino construido (2; 34).

Ant Colony System

El Sistema de Colonias de Hormigas es uno de los primeros algoritmos sucesores de AS. ACS incorpora tres modificaciones con respecto a AS (29):

- Aprovecha mejor la experiencia de búsqueda acumulada por las hormigas a través de una agresiva regla de selección.
- La evaporación y el depósito de feromona es aplicado solamente a las aristas que pertenezcan a la mejor solución global.
- Cuando una hormiga se mueva desde un nodo i hasta un nodo j a través de una arista $a(i,j)$, disminuye cierta cantidad de feromona de esa arista. Con esto se logra una mayor exploración de caminos.

La construcción de la solución en ACS se realiza una forma denominada regla proporcional pseudoaleatoria. Cuando una hormiga k se desea trasladar del nodo i hasta el siguiente nodo j la decisión es tomada a través de la siguiente distribución probabilística:

$$j = \begin{cases} argmax_{f \in N_i^k} \left\{ \tau_{if} [\eta_{if}]^{\beta} \right\}, si\ q \leq q_0 \\ J, \quad si\ q > q_0 \end{cases} \quad (8)$$

Donde q es una variable aleatoria uniformemente distribuida $[0,1]$; $q_0\ (0 < q_0 < 1)$ es un parámetro y J es una variable aleatoria seleccionada por la Ecuación 4.

La regla tiene como intención: cuando $q \leq q_0$ explotar el conocimiento, ya que utiliza los valores del rastro de feromona y la información heurística; y en cuando $q > q_0$ se realiza una exploración controlada, como se hacía en AS (1).

La actualización del rastro de feromona de forma global, se realiza después de cada iteración, es importante resaltar que la evaporación y el depósito de feromona solamente se le realiza a la mejor

hormiga hasta el momento, es decir a los arcos de la mejor solución siguiendo el valor devuelto por la siguiente ecuación:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij}^{bs} \quad (9)$$

Donde $\Delta\tau_{ij}^{bs} = {}^{1}/_{C^{bs}}$. ACS también permite la actualización del rastro de feromona de manera local. Cuando una hormiga incorpora un componente a su solución parcial, el rastro de feromona es actualizado con el valor devuelto por la ecuación:

$$\tau_{ij} \leftarrow (1-\varphi)\tau_{ij} + \varphi\tau_0 \quad (10)$$

Donde $\varphi \in (0,1]$ es la tasa local de evaporación de feromona y τ_0 es el valor inicial de feromonas. La regla *Online step-by-step pheromone*, incluye tanto la evaporación de feromona como el depósito de la misma. La cantidad de feromona depositada por cada hormiga es pequeña y la aplicación de esta regla, propicia que los rastros de feromona entre las aristas recorridas disminuyan. De esta manera ACS logra una manera de exploración adicional debido a que las aristas atravesadas por un gran número de hormigas sean menos atractivas para el resto que recorren en la iteración actual, logrando así una mayor diversificación del problema actual (1; 31).

MAX-MIN Ant System

El algoritmo *MAX-MIN Ant System* (35; 34) produce las siguientes modificaciones con respecto a AS:

- La hormiga que construyó la mejor solución en una iteración o la mejor solución global, es la única que actualiza los rastros de feromona.
- Se acota la cantidad de feromona que puede tener una arista determinada por un intervalo $[\tau_{max}, \tau_{min}]$, de esta manera es contrarrestada la acción de que únicamente la mejor hormiga deposite feromona ya que esta acción provocaría que todas las hormigas convergieran en una misma solución sin evaluar otras posibles óptimas soluciones.
- La inicialización de los rastros de feromona se realizan con el mayor valor posible (τ_{max}), así junto con un pequeña tasa de evaporación se logra una mayor exploración en el inicio de la búsqueda.
- El algoritmo cuando se produce un estancamiento de la búsqueda conducido por el sistema o no se encuentran mejores soluciones después de cierto número de iteraciones se produce lo que se conoce como reinicialización del rastro de feromona.

Cuando todas las hormigas han construido su solución, se actualizan los valores de feromona, primero aplicando una evaporación del rastro de feromona en correspondencia con la ecuación (5). Dando paso al depósito de feromona por la hormiga con la mejor solución

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{bs} \quad (11)$$

Donde $\Delta\tau_{ij}^{bs}$ es la cantidad de feromona que la mejor hormiga deposita en las aristas pertenecientes a su solución, definida como:

$$\Delta\tau_{ij}^{bs} = \begin{cases} {}^{1}/_{C^{bs}}, & si\ a_{ij} \in L^{bs} \\ 0, & en\ otro\ caso \end{cases} \quad (12)$$

Donde C^{bs} es el costo de la mejor solución global. También se le permite añadir feromona a la hormiga que generó la mejor solución en la iteración, es ese caso $C^{bs} = C^{bi}$, donde C^{bi}es el costo de la mejor solución generada por una hormiga en una iteración. Según experimentos realizados demuestran que se obtiene mejores rendimientos al aumentar gradualmente la frecuencia de la selección de la mejor solución global para incrementar el rastro de feromona (36).

Los límites inferior y superior ($\tau_{min}\ y\ \tau_{max}$) sobre los posibles rastros de feromona en cualquier arista, son otorgados para evitar estancamiento de la búsqueda y propiciarle a cada conexión, aunque bastante pequeña, la probabilidad de que sean escogidas. Después de cada iteración se debe asegurar que el rastro de feromona está en el umbral, por lo cual si $\tau_{ij} > \tau_{max}$, se ajusta $\tau_{ij} = \tau_{max}$, y si $\tau_{ij} < \tau_{min}$ se ajusta $\tau_{ij} = \tau_{min}$. En particular los límites impuestos tienen el efecto de limitar indirectamente la probabilidad (p_{ij}) de seleccionar un nodo j estando la hormiga en el nodo i al intervalo $[p_{min}, p_{max}]$, con $(0 < p_{min} \leq p_{ij} \leq p_{max} \leq 1)$. Sólo si una hormiga tiene una sola opción posible de moverse al siguiente nodo tal que $|N_i^k| = 1$, entonces ocurre el caso que $p_{min} = p_{max} = 1$.

Para estimar el valor de τ_{max} se toma como referencia el mayor valor que pudiera tener el rastro de feromona $\tau_{max} = {}^{1}/_{\rho C^{*}}$, donde C^{*} es el costo óptimo. Como no se dispone C^{*}, se utiliza el costo de la mejor solución global C^{bs} encontrada hasta el momento .Por tal motivo en cada iteración que se obtenga una mejor solución se debe ajustar el valor de τ_{max}. Para τ_{min} sólo es necesario escoger un valor constante que sea menor que τ_{max}.

Para la inicialización de los rastros de feromona al comienzo del algoritmo, es usado un valor estimado del límite superior del rastro de feromona. Estimación que puede generarse a través de la construcción de una solución L^* (mediante cualquier heurística *greddy*) y sustituyéndola en la ecuación de τ_{max} .En combinación con un valor pequeño para el factor de evaporación, se logra aumentar la exploración en las primeras etapas del algoritmo (26).

MMAS introduce un medio de exploración adicional, para los caminos que tienen una baja probabilidad de ser elegidos, a través de la reinicialización de los rastros de feromona. Este mecanismo se activa cuando el algoritmo se acerca a una situación de estancamiento (medido por estadísticas) o si no se han encontrado mejores soluciones por un determinado número de iteraciones.

1.5 Soluciones encontradas

A nivel internacional se han realizado diferentes aplicaciones basadas en la metaheurística ACO. A continuación se presentan dos ejemplos de estas:

- *Java Ant Colony System Framework* (según siglas en Inglés JACSF): Es un *framework* orientado a objetos escrito en Java. JACSF proporciona un esqueleto básico con un alto nivel de abstracción por el cual se puede implementar y resolver problemas a través de la técnica *Ant Colony System* (37).
- Libaco: Es una biblioteca creada como resultado del trabajo de Diploma en Ingeniería de Thomas Hammerl, estudiante de la Universidad Tecnológica de Viena (38). Libaco se basa en la búsqueda de solución de a problemas de NP-Completos a través de la implementación de las técnicas *Ant System*, *Elitist Ant System*, *Ant Colony System*, *Max-Min Ant System* y *Rank-Based Ant System*. Libaco está implementada en el lenguaje de programación C++ (38).

Para la presente investigación se necesita una biblioteca de propósito general implementada en Java, la cual contenga varios algoritmos de Optimización basada en Colonias de Hormigas y sea de fácil integración con la plataforma T-Arenal. Las soluciones encontradas no pueden ser utilizadas por:

- La solución JACSF, no es utilizada debido a que posee implementada una técnica de ACO y para poder solucionar un problema con ella es necesario implementar gran parte del funcionamiento del algoritmo ACS.

- La biblioteca Libaco contiene cinco técnicas de ACO pero no es utilizada porque está implementada en el lenguaje de programación C++ y debido a esto no es posible la integración con la plataforma T-arenal.

1.6 Introducción a la programación paralela y distribuida

En la actualidad se observa que en el paradigma orientado a objetos, solamente se puede ejecutar un equipo a la vez como máximo, en cambio con la introducción de las hebras concurrentes (programación concurrente) o procesos concurrentes, es posible que cada objeto se ejecute simultáneamente, esperando mensajes y respondiendo adecuadamente. Como siempre la principal razón para la investigación de la programación concurrente es que ofrece una manera diferente de conceptualizar la solución de un problema, una segunda razón es la de aprovechar el paralelismo del hardware subyacente para lograr una aceleración significativa.

1.6.1 Arquitecturas paralelas

La clasificación tradicional de Michael Flynn (39) de las arquitecturas paralelas se basa en dos criterios: el número de secuencias de instrucciones y el número de flujos de datos que definen las siguientes cuatro clases (6):

- **SISD** (*Single Instruction Stream—Single Data Stream*): Representa la arquitectura monoprocesador tradicional en la ejecución de un programa secuencial. Esta clase tiende a desaparecer. Hoy en día, la mayoría de los procesadores que componen las estaciones de trabajo y ordenadores portátiles son procesadores multinúcleo.
- **SIMD** (*Single Instruction Stream—Multiple Data Streams*): Son arquitecturas paralelas donde se ejecuta el mismo flujo de instrucción pero en datos diferentes. Los procesadores están restringidos a ejecutar el mismo programa. Tiene un modelo de programación sincrónica que se basa en la descomposición de datos (paralelismo de datos). Son muy eficientes en la ejecución de algoritmos paralelos sincronizados que contienen cálculos regulares y transferencias de datos regulares.
- **MISD** (*Multiple Instruction Streams—Single Data Stream*): Permite ejecutar múltiples flujos de instrucciones en un único grupo de datos. Existen muy pocos ejemplos donde se utilice esta arquitectura.
- **MIMD** (*Multiple Instruction Streams—Multiple Data Streams*): Se caracterizan por permitir que múltiples procesadores ejecutan en cualquier instante de tiempo distintas instrucciones sobre distintos datos. El control se realiza en forma descentralizada por cada uno de los procesadores que tienen autonomía de procesamiento. En este grupo se encuentran los

sistemas distribuidos o también conocidos como computadoras paralelas débilmente acopladas.

1.6.2 Introducción a los sistemas distribuidos

Un sistema distribuido (SD) constituye un grupo de computadoras autónomas integradas mediante una red, de preferencia por canales de alta velocidad, que se muestran ante los usuarios como una única máquina para darle solución a un problema determinado (40; 41).

En el caso de un sistema de cómputo distribuido la problemática a resolver en la mayoría de las ocasiones demanda grandes cálculos, por lo que se debe tratar de reducir el tiempo de obtención de la respuesta, esto se logra distribuyendo los cómputos, de forma transparente para el usuario, entre diversas computadoras.

Muchas definiciones con relación a este tema se encuentran en la literatura. Según Tanenbaum (42) un SD es una colección de computadoras autónomas que se muestran de cara al usuario del sistema como una única computadora. Según Puder (43) un SD es un sistema para el procesamiento de información compuesto por distintas computadoras independientes que concurren entre ellas mediante una red de comunicaciones con el objetivo de alcanzar un objetivo específico.

1.6.3 Modelos de distribución

Modelo Maestro-Esclavo

Es un paradigma que consiste en tener un proceso maestro (o varios) el cual debe definir trabajos y asignarlos a los procesos esclavos, que a su vez estos últimos generan soluciones y devuelven la mejor solución encontrada por ellos (óptima local). El proceso maestro es el encargado de valorar las soluciones devueltas por cada esclavo y tomar la mejor de todas, la cual sería la solución óptima global. Existen diferentes formas de asignación de tareas que están en dependencia del tamaño de las mismas. Con la utilización de este modelo se debe tener en cuenta que se puede producir en el maestro un cuello de botella, lo cual puede ocurrir cuando las tareas sean demasiado pequeñas y los esclavos demasiado rápidos. La granularidad[3] se debe elegir en correspondencia entre los costos de la realización del trabajo, la transferencia y sincronización de los procesos (44).

Modelo de islas

[3] En la computación paralela y distribuida es el equivalente a la razón cómputo-comunicación, es el tamaño de las piezas en que se divide una aplicación.

La población de divide en varias subpoblaciones y son distribuidas por los diferentes islas (nodos), las cuales son los responsables de la ejecución y solución de las tareas asignadas. Cada isla podrá utilizar diferentes valores de parámetros así como diferentes estrategias de solución, por lo cual se obtendrán soluciones diferentes. Al concluir todas las ejecuciones todas las poblaciones de las islas son combinadas con el propósito de construir una población global de la cual se seleccionará la mejor solución. Este modelo inserta como concepto novedoso la migración, la cual consiste en el intercambio sistemático de poblaciones entre las islas de acuerdo a la topología de interconexión utilizada (44).

Topologías de interconexión (45)

- **Estrella**: esta topología cuenta con una subpoblación que es designada como maestra (la que posee mejor media en el valor de la función objetivo), y las otras subpoblaciones son consideras esclavas. Estas últimas mandan sus mejores individuos a la subpoblación maestra, la cual envía a su vez sus mejores individuos a cada una de las subpoblaciones esclavas.
- **Anillo**: cada subpoblación envía sus mejores soluciones (o individuos) a una subpoblación vecina. En esta topología se puede realizar el flujo de migración de forma direccional (en un solo sentido) y bidireccional (en dos sentidos).

1.6.4 Medidas de evaluación de desempeño

La evaluación de desempeño se puede realizar de forma teórica y experimental. La comunidad científica ha propuesto una gran variedad de métricas para evaluar distintos aspectos del desempeño entre los cuales se encuentran *speed-up* y eficiencia.

Ganancia de velocidad (*Speed-Up*)

El ***Speed-Up*** $(\boldsymbol{S_P})$**,** vincula el tiempo de ejecución de un programa secuencia T_S y el tiempo total de ejecución de la versión en paralelo T_P de dicho programa ejecutado sobre P cantidad procesadores, como se muestra en la ecuación (46):

$$S_P = \frac{T_S}{T_P} \quad (13)$$

Esta métrica mide la ganancia de velocidad que se ha obtenido con la ejecución en paralelo, respecto al algoritmo secuencial. El tiempo de ejecución de un programa en paralelo con P procesadores, será P veces inferior al de su ejecución secuencial (un solo procesador), con el mismo poder de cómputo. En la práctica existen restricciones que imposibilitan la concreción del

objetivo planteado, como son las demoras en las comunicaciones, el intercambio de datos y la sincronización de los procesos.

Eficiencia $(\boldsymbol{E})$

La eficiencia significa el grado de aprovechamiento de los procesadores para la resolución del problema. Los valores de eficiencia deben estar entre 0 y 1, siendo los valores cercanos a 1 los más deseables (46).

$$E = \frac{S_P}{P} \quad (14)$$

Donde P significa la cantidad de procesadores utilizados.

1.7 Optimización basada en Colonias de Hormigas en Entorno Distribuidos

Los algoritmos de Optimización basados en Colonias de Hormigas debido a su estructura son muy buenos candidatos para ser paralelizados, pero no existen demasiados estudios sobre el tema El estudio sistemático de la aplicación de técnicas de programación paralela sobre ACO es reciente y varios autores han coincidido en que no se ha realizado un trabajo exhaustivo como para otras metaheurísticas (46).

1.7.1 Taxonomía de paralelismo aplicado a ACO

A continuación se presentan algunas consideraciones realizadas por autores referentes a la aplicación de paralelismo a ACO:

1. Según el trabajo de Randall y Lewis (47) incluye una de los pocas definiciones de estrategias de paralelismo aplicado a ACO existentes. Los autores definen cinco categorías:

- **Colonias de hormigas paralelas independientes** (*Parallel independent ant colonies*): corresponde a la ejecución en paralelo de un algoritmo ACO secuencial sobre un conjunto de procesadores disponibles. Las ejecuciones son completamente independientes, sin existir comunicación entre las colonias, que pueden utilizar los mismos o distintos parámetros.
- **Colonias de hormigas paralelas con interacción** (*parallel interacting ant colonies*): corresponde a la ejecución en paralelo de un algoritmo ACO secuencial sobre un conjunto de procesadores disponibles. Con una frecuencia fija de iteraciones, las colonias sincronizan sus matrices de feromona, obteniendo los valores de la mejor colonia hasta el momento. El volumen de datos que es necesario transmitir entre las colonias es alto ya que debe comunicarse la matriz de feromona completa.

- **Evaluación en paralelo de componentes de soluciones** *(parallel evaluation of solution elements):* corresponde a la ejecución de una colonia en forma secuencial, en la que cada hormiga evalúa en forma paralela las componentes que puede incorporar a la solución que está construyendo. Se utiliza un modelo maestro-esclavo, siendo el maestro la hormiga y los esclavos los encargados de evaluar las componentes. Randall y Lewis señalan que este tipo de estrategias puede ser recomendable para problemas con fuertes restricciones.
- **Hormigas paralelas** *(parallel ants):* corresponde con un modelo maestro-esclavo que trabaja sobre una colonia en forma paralela. El proceso maestro se encarga del manejo global de la matriz de feromona y cada proceso esclavo construye una solución como si se tratara de una hormiga de la colonia. La comunicación involucra el envió de las soluciones (o la actualización correspondiente de la matriz de feromona) desde los esclavos al maestro y el envió de la actualización de la matriz de feromona desde el maestro a los esclavos.
- **Combinación paralela de hormigas y evaluación de componentes de soluciones** (*parallel combination of ants and evaluation of solution elements*): corresponde a la combinación de las estrategias hormigas paralelas y evaluación en paralelo de componentes de soluciones. El modelo implica tener dos niveles de paralelismo maestro-esclavo, uno a nivel de la colonia y las hormigas, y otro a nivel de cada hormiga y las evaluaciones de las componentes de las soluciones.

2. Se han presentado varios aspectos relevantes para la implementación de los mismos al respecto (5), entre los cuales se destaca, que la creación de la solución por parte de una hormiga es un proceso secuencial y no suele ser separados por varios procesadores, por lo que la granularidad mínima del algoritmo suele ser la construcción de una solución. Se abordan especificaciones del tema del modelo multicolonias, el que consiste en poseer un conjunto de colonias de hormigas que usan sus propias matrices de feromonas. Este modelo tiene como objetivo aumentar la calidad de las soluciones. Los autores definen un conjunto de aspectos que deben ser considerados para crear un algoritmo ACO multicolonias (5). Los cuales son:

- **Estructura de comunicación y topologías de la vecindad:** Se establece la estructura de comunicación y la estructura de vecindad entre las colonias. Se utilizan las estructuras de anillo, hypercube, completa y aleatoria.
- **Tipo de información intercambiada entre colonias:** Las informaciones intercambiadas entre las colonias pueden ser soluciones (que puede ser una hormiga, la mejor solución global de las colonias, la mejor solución en una vecindad, la mejor solución de una colonia enviada al vecindario), vectores de feromona y la matriz de feromona.

- **Forma de utilización de la información recibida por otras colonias:** Las opciones generalmente utilizadas son: sustituir una solución elite cuando la solución recibida es mejor, actualizar la matriz de feromona con la solución recibida o agregar a la generación actual la solución recibida.
- **Frecuencia de comunicación:** Entre las variantes comunicación se encuentran: comunicación en cada iteración, comunicación con una frecuencia fija de iteraciones y comunicación dependiente de la calidad de las soluciones obtenidas.
- **Enfoque homogéneo frente al planteamiento heterogéneo:** Es posible usar dentro de una misma iteración diferentes métodos para construir las soluciones y/o actualizar el rastro de feromona. También es posible usar diferentes estrategias entre iteraciones.

1.8 Herramientas utilizadas para la solución

Seguidamente se abordan elementos sobre las herramientas seleccionadas para llevar a cabo la propuesta de solución al problema a resolver planteado en esta investigación.

1.8.1 Herramientas CASE

Ingeniería de Software Asistida por Ordenador (CASE): constituyen un conjunto de programas y ayudas que aportan asistencia a los analistas, ingenieros de software y desarrolladores, durante la totalidad de los pasos del ciclo de vida del desarrollo de un software (48).

Actualmente existen gran variedad de herramientas *CASE*. Entre las mejores y más utilizadas se encuentran *EasyCASE, Oracle Designer, System Architect* y *Visual Paradigm for UML*. Para la modelación de esta propuesta de solución se utilizará dicha herramienta en su versión para UML 8.0 *Enterprise Edition*, debido a que la UCI posee su licencia de uso.

Visual Paradigm for UML 8.0 Enterprise Edition

Herramienta profesional que da soporte al modelado visual mediante UML[4], es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. Soporta todo el ciclo vital del desarrollo de software. Considerada muy completa y con suficientes funcionalidades, lo que contribuye a la creación rápida de aplicaciones de calidad. Permite modelar todos los tipos de diagramas de clases, generar código desde diagramas, posee interoperabilidad con otros sistemas e integración con distintos Ambientes de Desarrollo Integrado (IDE), provee abundantes tutoriales y crea de manera simple toda la documentación, pudiendo ser salvada en formatos tales como PDF y HTML (49).

Algunas de las ventajas que posee esta herramienta son las siguientes:

[4] UML: Unified Modeling Lenguage por sus siglas en inglés (Lenguaje de Modelado Unificado).

- Disponibilidad en múltiples plataformas.
- Oportunidad de intercambiar información mediante la importación y exportación de ficheros con aplicaciones.
- Posee apoyo adicional en cuanto a generación de artefactos automáticamente.
- Ofrece la posibilidad de generar código a partir de los diagramas y viceversa.
- Posibilita la creación de documentación sin necesidad de utilizar herramientas externas.

1.8.2 Lenguaje de Programación Java

Un lenguaje de programación se define como un lenguaje para describir el conjunto de acciones consecutivas que un equipo debe hacer. De esta forma, constituye un modo práctico para que los seres humanos puedan dar instrucciones a un computador. Dicho lenguaje se compone por una serie de reglas gramaticales, símbolos utilizables, términos con sentido único y una regla principal que resume las demás (50).

Java es un lenguaje de programación que con el que resulta de alguna forma sencillo el escribir instrucciones, implementar algoritmos o diversas funcionalidades o componentes, debido a que posee un conjunto de características que facilitan su empleo. Dichas características propias de ese lenguaje son:

- **Orientado a objetos**: Diseñado como un lenguaje orientado a objetos desde su creación. Los objetos concentran en estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manejan o gestionan esos datos.
- **Distribuido**: Provee una colección de clases para el uso de aplicaciones para red, que permiten acceder a *sockets* e implantar y aceptar conexiones con servidores o clientes remotos, facilitando de esta forma la implementación de aplicaciones distribuidas.
- **Robusto**: Implementado para construir *software* fiable en gran medida. Para ello aporta cuantiosas comprobaciones en compilación y en tiempo de ejecución.
- **Portable**: Creado con indiferencia a la arquitectura, especificación de los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas creados en este lenguaje son iguales sin importar la plataforma.

1.8.3 Entorno de Desarrollo Integrado (IDE)

NetBeans IDE 6.9

Constituye un entorno de desarrollo para programadores creado por *Sun MicroSystems*[5], que posibilita escribir, compilar, depurar y ejecutar programas. Se encuentra implementado para Java, pero puede aprovecharse correctamente para cualquier otro lenguaje de programación. Contiene además número importante de módulos para su extensión.

Entre sus principales características se encuentran:

- Soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles).
- Soporta el desarrollo de herramientas de desarrollo visuales de SOA, herramientas de esquemas XML, orientación a web servicies (BPEL), y modelado UML.
- Modularidad. Todas las funciones del IDE son provistas por módulos.
- Permite crear aplicaciones web con PHP 5, *AJAX*, *Python*, cuenta con un potente *debugger*[6] integrado y además viene con soporte para *Symfony*.

1.8.4 Plataforma de Tareas Distribuidas T-arenal

La Plataforma de Tareas Distribuidas (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”. T-arenal v2.0 fue implementada con una arquitectura de varios servidores para una mejor configuración y uso más equitativo de los clientes. 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 (51).

T-arenal v2.0 está basada en el modelo Cliente - Servidor. Está dividida en tres componentes esenciales: servidor central, servidor de peticiones y cliente, como se muestra en la figura

[5] Empresa productora de software a nivel mundial.
[6] Funcionalidad que permite seguir paso a paso la ejecución de un algoritmo.

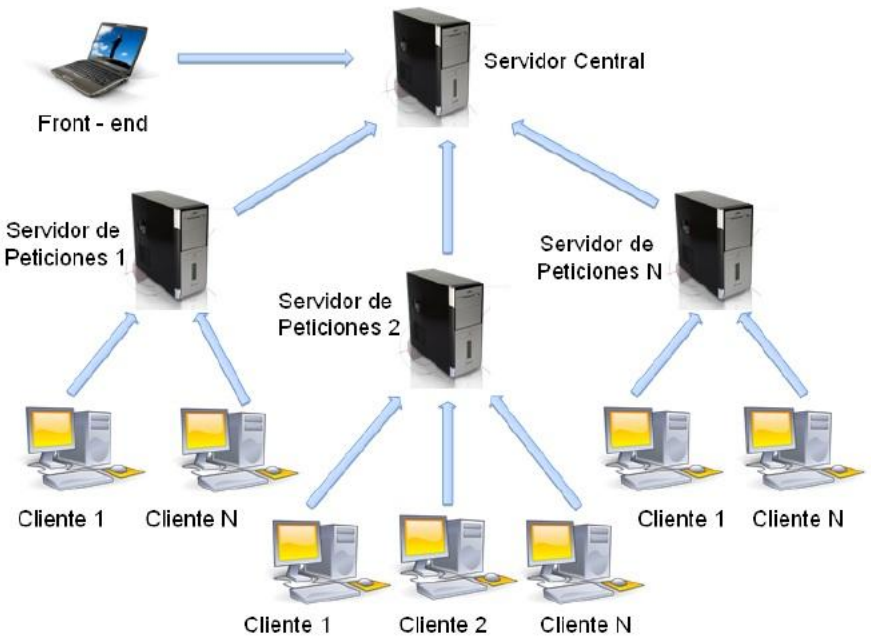


Figura 5: Vista general de la Plataforma de Tareas Distribuidas v2.0.

Mediante el servidor central se realiza toda la gestión de los grupos, usuarios, rangos de IP, clientes, problemas, ejecuciones y soluciones del sistema. Al servidor central estarán asociados uno o varios servidores de peticiones los cuales son los encargados de solicitar una tarea. A cada servidor de peticiones le pertenecerán uno o varios clientes para realizar el procesamiento de una tarea determinada. El objetivo de un servidor de peticiones es repartir el trabajo entre los clientes correspondientes, gestionar la pérdida de clientes y conformar la solución final a partir de los resultados enviados por cada cliente.

1.9 Conclusiones

En el capítulo se abordaron conceptos que sirvieron de base para alcanzar un adecuado dominio de la presente investigación. Se trataron temas referentes a la metaheurística ACO, así como, otros conceptos con el objetivo de alcanzar una mejor comprensión de la misma. Se definieron las tecnologías y herramientas que se utilizarán para alcanzar el desarrollo adecuado y exitoso de la biblioteca.

CAPÍTULO 2: DISEÑO E IMPLEMENTACIÓN DE LA BIBLIOTECA ACO

En este capítulo se abordan los temas relacionados con el diseño e implementación de la biblioteca ACO. Se describen las principales clases e interfaces de la biblioteca. También se explica el funcionamiento básico de la misma y su forma de utilización.

2.1 Diseño de la Biblioteca ACO

Esta biblioteca está diseñada de forma genérica, lo cual posibilita una fácil adaptación para resolver una gran diversidad de problemas de optimización. Para la utilización de la biblioteca es necesario adaptar cada problema para darle solución, esto es alcanzado con la introducción por parte del usuario de una serie de datos específicos en cada problema. La biblioteca brinda la posibilidad de ser utilizada de forma secuencial y/o distribuida sobre la plataforma T-arenal. En la Figura 6 se muestra el diseño de clases de la biblioteca.

La concepción de la biblioteca está basada en el paradigma de la Programación Orientada a Objetos (POO), permitiendo las siguientes ventajas:

- **Reusabilidad**: una misma implementación de un algoritmo permite solucionar diferentes problemas
- **Facilidad de aprendizaje y uso**: el usuario simplemente tiene que completar las clases que necesita para la resolución del problema, sin tener la necesidad de conocer la implementación del algoritmo.
- **Facilidad de ampliación**: la biblioteca cuenta con varios mecanismos de ampliación como son el polimorfismo, la herencia y la redefinición de código.

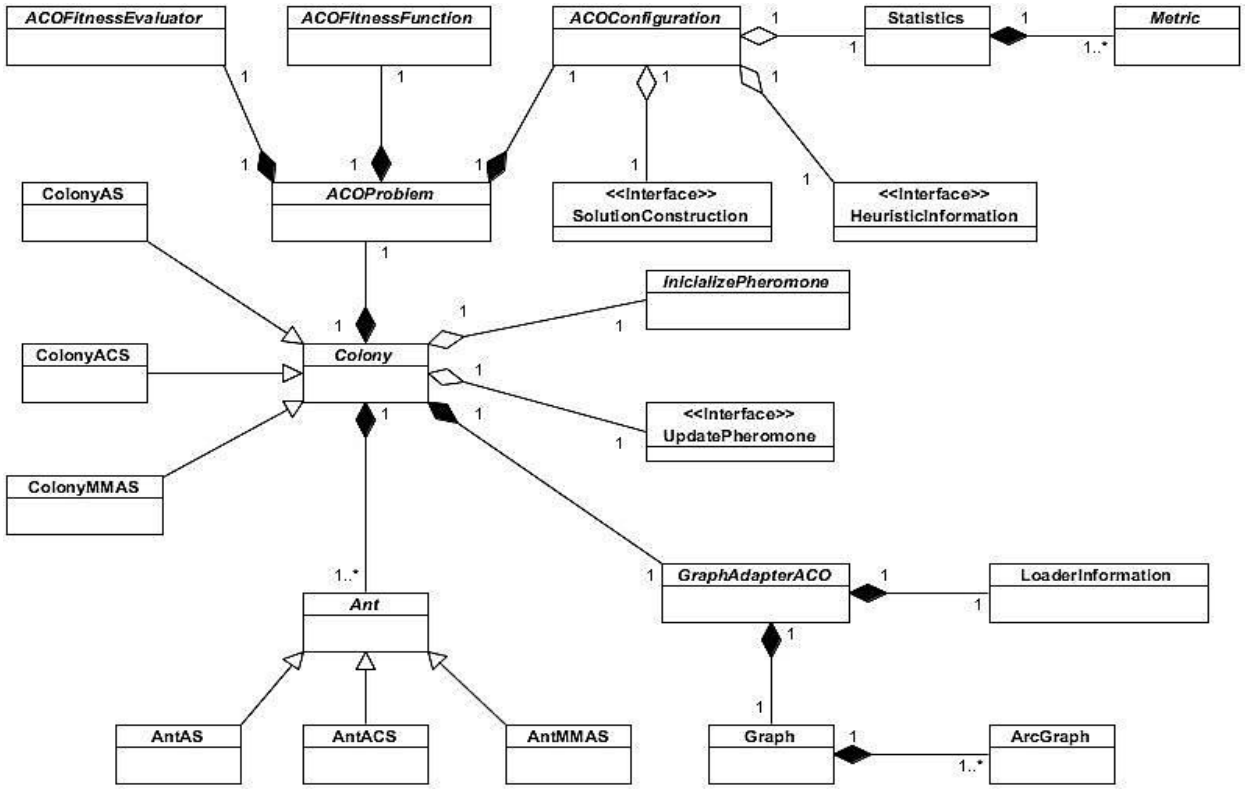


Figura 6: Diagrama UML de la biblioteca ACO.

Funcionamiento lógico de la biblioteca ACO

Para la utilización de la biblioteca por parte del usuario, este debe modelar el problema que desea resolver (*ACOProblem*) y definir la configuración de los parámetros (*ACOConfiguration*), así como, la manera en que cada hormiga artificial (*Ant*) construirá una solución del problema (*SolutionConstruction*), la información heurística (*HeuristicInformation*), la cual será utilizada por las hormigas para elaborar las soluciones y la función objetivo (*ACOFitnessFunction*) del problema a resolver. También el usuario debe definir el grafo de construcción (*GraphAdapterACO*) por el cual las hormigas se moverán. Otro aspecto a tener en cuenta son las métricas (*Metric*) que se deseen evaluar, las cuales serán precisadas (*Statistics*) por parte del usuario antes de ejecutar el algoritmo.

Después de haber configurado los parámetros de la biblioteca y de seleccionar cuales de las técnicas de ACO se desea utilizar (ColonyAS, ColonyACS o ColonyMMAS), el algoritmo está listo para comenzar con el proceso de resolución del problema. Cuando el usuario ordena la ejecución del algoritmo, este último inicializa todos los parámetros (tamaño de la población, cantidad de iteraciones, condiciones de parada, inicialización de la matriz de feromona (*InicializePheromone*) y la forma en que se actualizará (UpdatePheromone), entre otros) a partir de la información brindada por el problema.

Luego en cada iteración que el algoritmo realiza, siempre y cuando no se cumpla ninguna de las condiciones de parada, se selecciona la mejor solución generada entre todas las hormigas y se actualiza la matriz de feromona.

2.2 Implementación de la biblioteca ACO

A continuación se muestran las principales clases e interfaces de la biblioteca.

2.2.1 Clases de la Biblioteca ACO

Clase ACOProblem

En esta clase se representa el problema de optimización que se desea resolver. Cada problema a solucionar debe expresar sus especificidades en esta clase.

Esta clase cuenta entre sus métodos con:

- *boolean checkSolution(Ant ant)*: es un método abstracto que debe ser implementado por el usuario. Esta funcionalidad se encarga de verificar si la solución que la hormiga está creando es válida, en dependencia de las condiciones y parámetros que sean definidas.

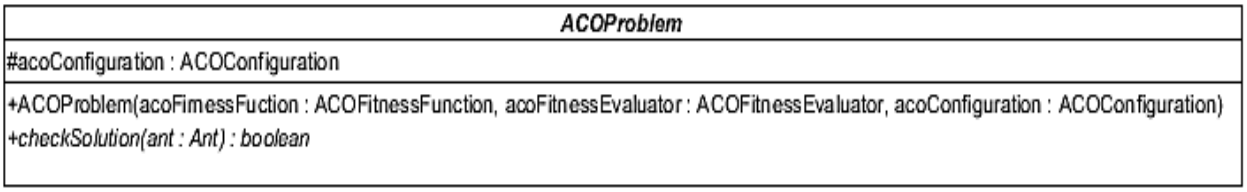


Figura 7: Diagrama UML de la clase *ACOProblem*.

Clase Colony

Esta es la clase abstracta encargada de gestionar todo el funcionamiento del algoritmo. Gestiona las soluciones que generan las instancias de la clase *Ant* y selecciona la solución más óptima para el problema planteado. Esta clase crea la colonia de hormigas artificiales con el tamaño poblacional definido para el problema en cuestión, así como, de realizar tantas iteraciones del algoritmo como se hallan defino con antelación. Aquí también se evalúan las métricas definidas por el usuario a través del atribute *statistics* y se realiza la inicialización de los valores de la matriz de feromona (*initializePheromone*).La biblioteca tiene implementada tres técnicas ACO, cada una de estas heredan de esta clase (*ColonyAS*, *ColonyACS* y *ColonyMMAS*).

Esta clase posee como métodos fundamentales los que se presentan a continuación:

- *void performInitilisation()*: se encarga de la inicialización de todas las instancias de las hormigas artificiales que conforman la colonia, así como, de los valores necesarios para el funcionamiento del algoritmo.
- *void performIteration():* se encarga de realizar las iteraciones antes definidas para el problema a solucionar y selecciona en cada iteración la hormiga que generó la mejor solución (óptima global).
- *void run()*: se encarga de ejecutar el algoritmo. La ejecución concluye cuando se cumpla alguna de las condiciones de parada establecidas con anterioridad.
- *void addStoppingCondition(StoppingCondition stoppingCondition)*: se encarga de adicionar condiciones de paradas a una lista. Estas condiciones son las encargadas de detener la ejecución del algoritmo si alguna se cumple.
- *void isFinished()*: se encarga de determinar si se ha cumplido alguna condición de parada, y de esta forma se termina la ejecución del algoritmo.

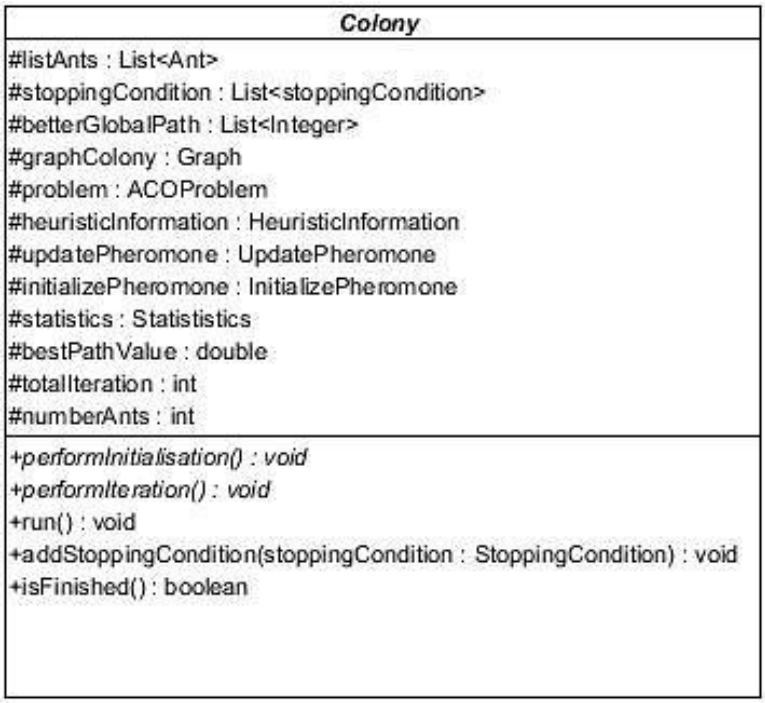


Figura 8 Diagrama UML de la clase *Colony*.

Clase Ant

Esta clase es el elemento fundamental del algoritmo, es la clase que conforma la colonia de hormigas artificiales. Se encarga de construir las soluciones del problema definido por el usuario. En la construcción de la solución de un problema, la influencia de los valores tanto de la información

heurística como la información del camino de la feromona, esta determina por los valores *alpha* y *beta* respectivamente.

Las funcionalidades principales de esta clase son:

- *double probabilityNextNode(int currentNode, int nextNode)*: se encarga de calcular la probabilidad que existe de que la hormiga se mueva de la posición i (*currentNode*) a la posición j (*nextNode*), basadas en la función probabilística de transición (explicada en el capítulo anterior) para la construcción de la solución del problema.
- *int stateTransationRule(int initialNode)*: se encarga de determinar hacia cuál nodo se moverá la hormiga en la construcción de una solución del problema. Esta regla está determinada por la técnica de ACO que se utilice para solucionar el problema definido.
- *List<Integer> getVisitedNode()*: se encarga de devolver una lista con un conjunto de valores que representan los nodos que la hormiga ha visitado en la construcción de una solución.

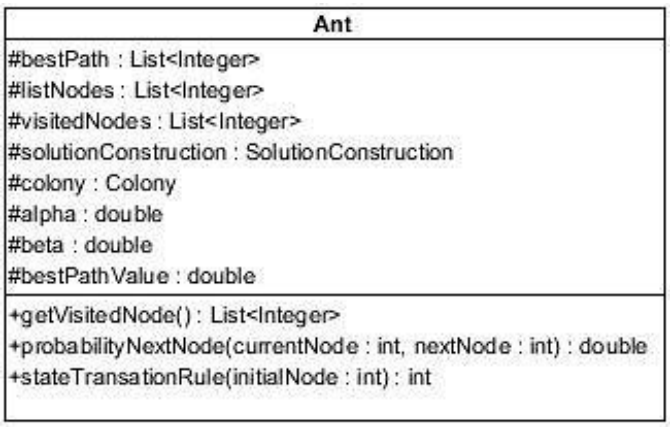


Figura 9 Diagrama UML de la clase *Ant*.

Clase Graph

Esta clase representa el grafo por el cual se desplazan las hormigas para la construcción de las soluciones (grafo de construcción). El grafo está compuesto por un arreglo bidimensional de la clase *ArcGraph*, la cual contiene dos tipos de información: el valor del arco (*arcInformation*) y el valor de la feromona (*pheromone*) que une dos nodos.

Entre los principales métodos de la clase se encuentran:

- *ArcGraph [][] getGraph()*: devuelve el arreglo bidimensional de la clase *ArcGraph* que representa el grafo necesario para la ejecución de los algoritmos ACO.

- *void addNodes()*: se encarga de adicionar los nodos o vértices que componen el grafo a una lista.
- *List<Integer> getListNodes()*: se encarga de devolver la lista de nodos que conforman el grafo.
- *Vector<double [][]> getAuxiliaryGraph()*: se encarga de retornar un vector compuesto por arreglos bidimensionales de matrices. Estas matrices conforman grafos que contienen información que puede necesitar el algoritmo para darle solución a un determinado problema. La utilización de este vector es opcional, está en correspondencia con las características del problema a resolver.
- *void showGraph()*: imprime por consola el grafo del algoritmo.

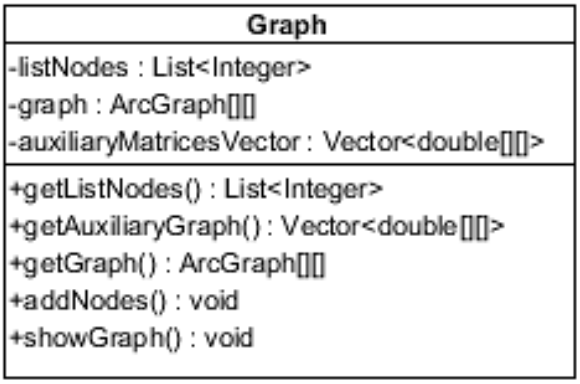


Figura 10 Diagrama UML de la clase *Graph*.

Clase GraphAdapterACO

Esta clase abstracta es utilizada en la biblioteca para que el usuario confeccione el grafo que utilizará el algoritmo para su ejecución. En esta clase se carga el fichero que contiene la información para la confección del grafo, a través de una instancia de la clase *LoadInformation*, la cual cuenta con el método *loadData*, este devuelve un vector con todas las matrices (grafos) que contienen la información que el problema necesita para su solución.

Los principales métodos de la clase son:

- *Graph graphAdapterACO()*: es un método abstracto, donde el usuario define el grafo que utilizará el algoritmo, basado en la información que brinda la instancia del problema en cuestión.

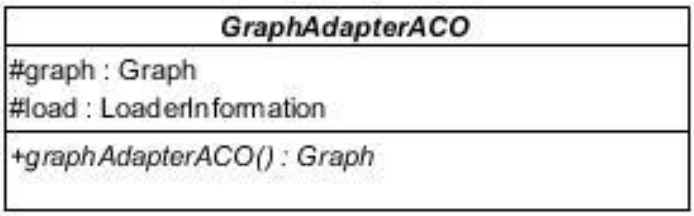


Figura 11 Diagrama UML de la clase *GraphAdapterACO*.

Clase ACOConfiguration

Esta clase se encarga de cargar los valores para configurar el algoritmo y poder solucionar el problema a resolver, como son por ejemplo, cantidad de hormigas de la colonia, cantidad de iteraciones que tendrá el algoritmo, el factor de evaporación que se utilizará, así como otros valores que pueden estar en dependencia de la técnica ACO que se utilice. Todos estos parámetros son definidos por el usuario en un fichero llamado con extensión *properties*. Toda esta información estará contenida en una instancia de la clase *Properties*, del lenguaje JAVA. También en la clase *ACOConfiguration* se deben definir cómo se construirá la solución del problema y la forma en que se evaluará la información heurística, a través de instancias de las clases *SolutionConstruction* y *HeuristicInformation* respectivamente.

Uno de los métodos más importante de esta clase es:

- *void initProperties(String direction)*: se encarga de cargar el fichero que contiene la información del problema a resolver y crea una instancia de la clase *Properties* con esta información.

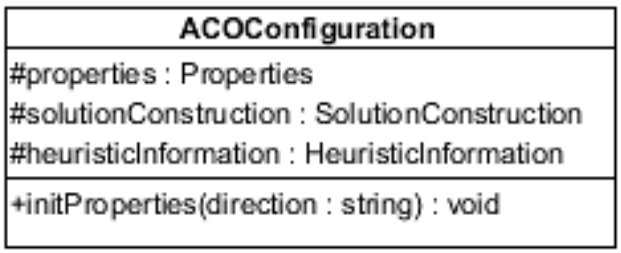


Figura 12 Diagrama UML de la clase *ACOConfiguration*.

Clase Metric

Es una clase abstracta, utilizada para definir las métricas que el usuario requiera para realizar el análisis del comportamiento del algoritmo y de los resultados arrojados por el mismo. Posee el método abstracto *evaluateMetric*, mediante el cual se precisa las valoraciones que se realizarán y el método *updateMetric*, almacena los datos obtenidos en cada iteración por las valoraciones antes mencionadas.

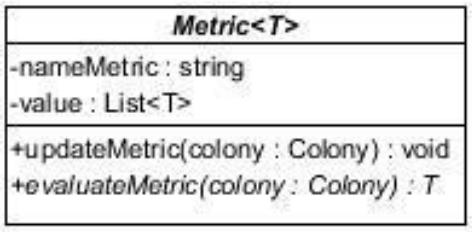


Figura 13 Diagrama UML de la clase *Metric*.

Clase Statistics

Esta clase de encarga de almacenar todas las métricas definidas por el usuario y evaluarlas en cada iteración a través de método *evaluateMetrics*. Esta clase posibilita almacenar en un fichero, seleccionado por el usuario, los valores de las métricas utilizadas a través del método *saveMetrics.*

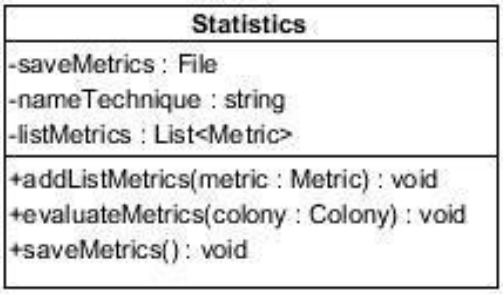


Figura 14 Diagrama UML de la clase *Statistics*.

Clase ACOFitnessFunction

A través de esta clase abstracta el usuario define la función objetivo del problema a resolver. Esta función objetivo permite valorar la calidad de la soluciones encontradas y de esta forma seleccionar la más adecuada para la solución del problema. El usuario en el método abstracto *calculateFitness* y utilizando la lista de nodos visitados (*visitedNodes*) de la clase *Ant*, la cual es pasada por parámetro, devuelve un valor de tipo *double*, que representa la calidad (fitness) de la solución construida por la hormiga.

Clase ACOFitnessEvaluator

Esta clase abstracta es la encargada de comparar las soluciones obtenidas en dependencia del objetivo del problema (si se está maximizando o minimizando). Posee dos métodos abstractos con el mismo nombre *isFitness*, pero con diferentes parámetros. A uno se le pasa dos valores *double*, y al otro se le pasa dos instancias de la clase *Ant*. Ambos métodos devuelven un valor *boolean*, en dependencia del resultado de la comparación.

2.2.2 Interfaces de la Biblioteca ACO

Interfaz HeuristicInformation

Implementando esta interfaz se define cómo se determinará la información heurística del problema en cuestión. La manera en que se conformará esta información se puntualiza cuando el usuario lo defina en el método *huristicInformation.* La biblioteca tiene implementado una variante de esta interfaz (*HeuristicInformationStandard*) y es utilizada en la biblioteca por defecto.

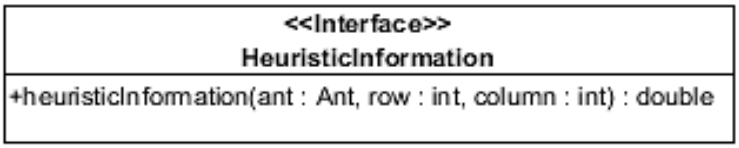


Figura 15 Diagrama UML de la interfaz *HeuristicInformation*.

Interfaz SolutionConstruction

Cada problema puede tener sus especificidades para la construcción de su solución. Implementando esta interfaz y definiendo el método *solutionConstruction*, se precisa la forma en que se construirá la solución del problema establecido por el usuario. La biblioteca brinda una implementación de esta clase (*SolutionConstructionStandard*) que es utilizada por defecto en la biblioteca, si es que el usuario no define una.

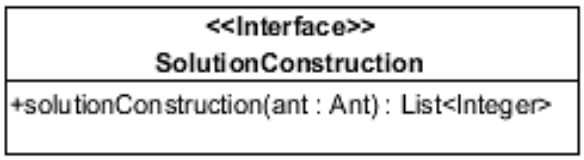


Figura 16 Diagrama UML de la interfaz *SolutionConstruction*.

Interfaz UpdatePheromone

Cada algoritmo de ACO puede actualizar la matriz de feromona de manera diferente, ya sea de forma local y/o de forma global. Implementando esta interfaz se puede realizar la actualización de la matriz de feromona de forma local implementando los métodos *localEvaporationPheromone* y *localReinforcementPheromone* y de forma global los métodos *globalEvaporationPheromone* y *globalReinforcementPheromone* para un determinado algoritmo.

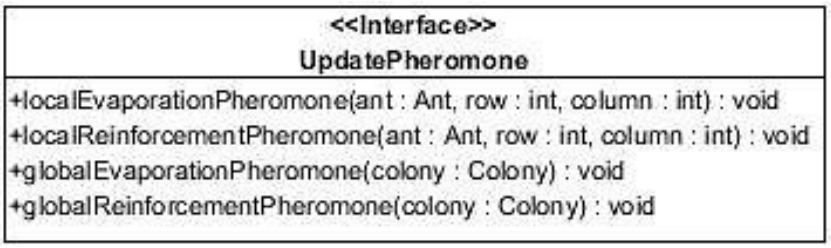


Figura 17 Diagrama UML de la interfaz *UpdatePheromone*.

2.3 Integración de la biblioteca ACO con la plataforma T-arenal

Para utilizar la biblioteca de forma distribuida sobre la plataforma T-arenal, el usuario debe heredar de dos clases: *DataManager* y *Task*. Para lograr este objetivo se deben definir unas funcionalidades de dichas clases

- **DataManager:** esta clase se utiliza en el servidor de peticiones. Es la encargada de componer los trabajos que ejecutaran las máquinas clientes a través de la implementación del método *generateWorkUnit*, así como, procesar los resultados devueltos por estos mediante el método *processResults*. También es la encargada de ajustar la granularidad de las estaciones de trabajo y chequear el cumplimiento de las tareas asignadas a los clientes.
- **Task:** esta clase se ejecuta en las estaciones de trabajo. En ella se encuentra la instancia del algoritmo que se ejecutará en las diferentes unidades clientes. Ella se encarga de devolver el o los resultados obtenidos después de haber realizado la ejecución de la tarea asignada a cada cliente. Esta funcionalidad se logra a través de la implementación del método *processUnit* por parte del usuario.

2.4 Conclusiones

En este capítulo se expuso el diseño e implementación de la biblioteca ACO, mediante la explicación de las principales clases e interfaces que la componen, así como los principales métodos que son necesarios para su correcto funcionamiento. Como principal resultado se obtuvo una biblioteca que cumple con las necesidades planteadas en la investigación.

CAPÍTULO 3: RESULTADOS Y DISCUSIÓN

En el presente capítulo se muestran y discuten los resultados alcanzados por la biblioteca en las pruebas realizadas mediante la modelación y resolución de un Problema de Asignación Cuadrática. Se efectúa una introducción a los Problemas de Asignación Cuadrática para lograr un mejor entendimiento del mismo. Se muestran los resultados de las pruebas realizadas, por la biblioteca, de forma secuencial y distribuida, valorando el parámetro tiempo de ejecución.

3.1 Problema de Asignación Cuadrática

El Problema de Asignación Cuadrática (QAP, por sus siglas en inglés, Quadratic Assignment Problem) fue introducido por Koopmans y Beckmann en 1957 como un modelo matemático para la ubicación de un conjunto de actividades económicas indivisibles (52). El QAP puede ser descrito mejor como el problema de asignar un conjunto de elementos a un conjunto de ubicaciones, donde hay distancias entre las ubicaciones y flujos entre los elementos. El objetivo de este problema eses situar los elementos en las ubicaciones de forma tal que la suma del producto entre flujos y distancias sea mínima.

Dado dos matrices de $n\ x\ n$, $A = (a_{ij})$ y $B = (b_{ij})$, donde a_{ij} representa distancia que existe entre las ubicaciones i y j y b_{ij} representa el flujo entre los elementos i y j; $\Phi_{(n)}$ es el conjunto de permutaciones de n elementos correspondientes a las asignaciones, pertenecientes al conjunto de enteros $\{1 \ldots . n\}$ y ϕ_i devuelve la ubicación del elemento i en la solución actual $\phi \in \Phi_{(n)}$ (53).

$$mín_{\phi \in \Phi_{(n)}} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{\phi i \phi j} b_{ij} \quad (15)$$

Se seleccionó el QAP para realizar las pruebas de la biblioteca debido a que es un problema de optimización NP- duro bien conocido y estudiado, para el cual no se conoce ningún algoritmo que lo resuelva en tiempo polinomial e incluso para instancias pequeñas se requiere de un gran tiempo computacional para su resolución (54; 55). Es considerado uno de los problemas de optimización combinatoria más difíciles de resolver en la práctica. Existen ejemplos prácticos como el Cableado de Tableros (56), Disposición de Campus y Hospitales (57), entre otros, que pueden ser formulados como instancias de QAP. Otros problemas de optimización combinatoria NP-duros, tales como el Problema del Viajante de Comercio (58)(TSP, del inglés, Traveling Salesman Problem) y el problema Bin-Packing (59) pueden ser modelados como QAP.

3.2 Aplicación de ACO al QAP

La representación del QAP en ACO (60) se lleva a cabo mediante un grafo de construcción G, donde el conjunto de vértices V representan los elementos y las ubicaciones de la descripción del problema, y el conjunto de arcos E conectan los vértices, es decir, hay arcos desde elementos a ubicaciones y arcos desde ubicaciones a elementos. La construcción de una asignación de elementos a ubicaciones (o de ubicaciones a elementos) puede ser interpretada como un movimiento de la hormiga sobre la representación del grafo de construcción del QAP, guiado por la información del rastro de feromona (asociado a los arcos) y posiblemente por información heurística disponible localmente.

El movimiento de la hormiga adicionalmente tiene que obedecer ciertas restricciones para asegurar que se genere una solución factible. En el caso del QAP, la solución factible consiste en asignar cada elemento a exactamente una ubicación y cada ubicación tiene que tener asignada exactamente un elemento. Por lo tanto, una solución factible φ para QAP consiste de n duplas (i, j) de elementos y ubicaciones. Es conveniente usar siempre una dirección de asignación fija, ya sea elementos a ubicaciones o viceversa.

Para la construcción de una solución, las hormigas realizan iterativamente los siguientes pasos: primero elegir un elemento y un segundo paso es asignar el elemento seleccionado a una ubicación. En el primer paso el rastro de feromona y la información heurística pueden ser utilizadas para determinar un orden de asignación y en el segundo paso el rastro de feromonas τ_{ij} y la información η_{ij} heurística asociadas a la dupla conformada por el elemento i y la asignación j determinan la deseabilidad de poner el elemento i en la ubicación j.

3.3 Modelado del problema QAP en la biblioteca

Para darle solución al QAP usando la biblioteca, se utilizaron clases definidas en ella y otras específicas. Se emplearon dos algoritmos basados en ACO los cuales se encuentran implementados en la biblioteca, estos son AS y MMAS.

Una solución construida por una hormiga para un QAP, en el modelo empleado, se define como una lista de permutaciones L de enteros, donde el elemento seleccionado se guarda en una posición L_i y la asignación de la ubicación, seleccionada por la hormiga, para ese elemento en el problema se almacena en la lista en la posición $L_{(i+1)}$, de esta manera se realiza la confección de la tupla compuesta por elemento-ubicación.

Las clases que se definieron para solucionar el QAP se presentan a continuación:

- **Información heurística:** *ACOHeuristicsInformationQAP*

- **Construcción de la solución:** *ACOSolutionConstructionQAP*
- **Configuración:** *ACOConfigurationQAP*
- **Problema:** *ACOProblemQAP*
- **Grafo de construcción:** *ACOGraphQAP*
- **Función objetivo:** *ACOFitnessFuntionsQAP*
- **Evaluador de la función objetivo:** *MinimizeFunction*

Los parámetros definidos para las pruebas realizadas son:

Población: 100

Factor de Evaporación: 0.1

Influencia del rastro de feromona (α): 1

Influencia de la información heurística (β): 1

Todas las pruebas se realizaron con la instancia: tai100b.dat. Esta instancia de QAP está disponible en el sitio QAPLib (61).

3.4 Evaluación de la biblioteca de manera secuencial

Las pruebas realizadas a la biblioteca se efectuaron en una PC con procesador Intel (R) Pentium (R) 4 a 3.00 GHz y memoria RAM de 1 GB y con sistema operativo Linux. Para las mismas se utilizaron los algoritmos AS y MMAS. Se utilizaron diferentes cantidades de iteraciones $I = \{50,100,250,500\}$, para observar el comportamiento de los algoritmos ante el incremento de la cantidad de trabajo manteniendo fijo el resto de los parámetros.

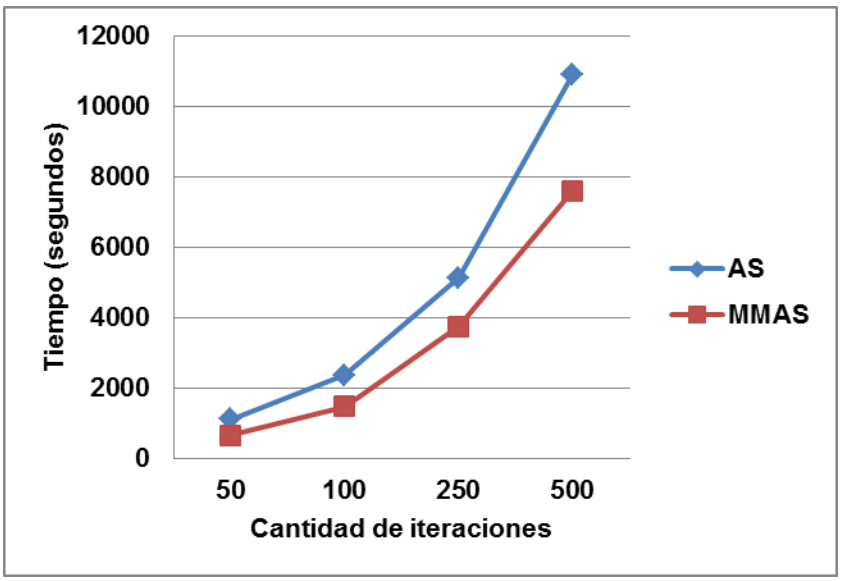
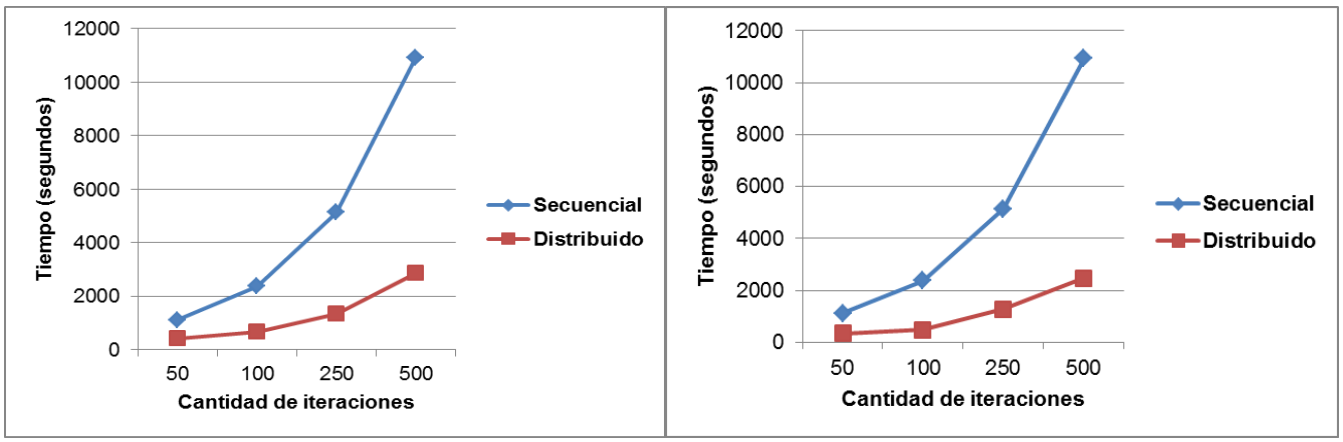


Figura 18 Comparación de tiempos de ejecución secuencial de los algoritmos AS y MMAS.

Puede observarse en la Figura 18 que en las ejecuciones, a medida que se incrementa la cantidad de iteraciones, aumenta gradualmente el tiempo de procesamiento. Se aprecia también que con las mismas configuraciones, el algoritmo MMAS tiende a tener una ejecución más eficiente en cuanto a tiempo que AS, esto se debe a que en el algoritmo AS la actualización de la matriz de feromona se realiza con las soluciones generadas por todas las hormigas en cada iteración, lo cual comparado con la actualización que realiza el algoritmo MMAS es mucho más lento, ya que este únicamente actualiza la matriz de feromona con la mejor solución global o la mejor solución en la iteración según sea definido por el usuario (ver epígrafe 1.4.4).

3.5 Evaluación de la biblioteca de manera distribuida

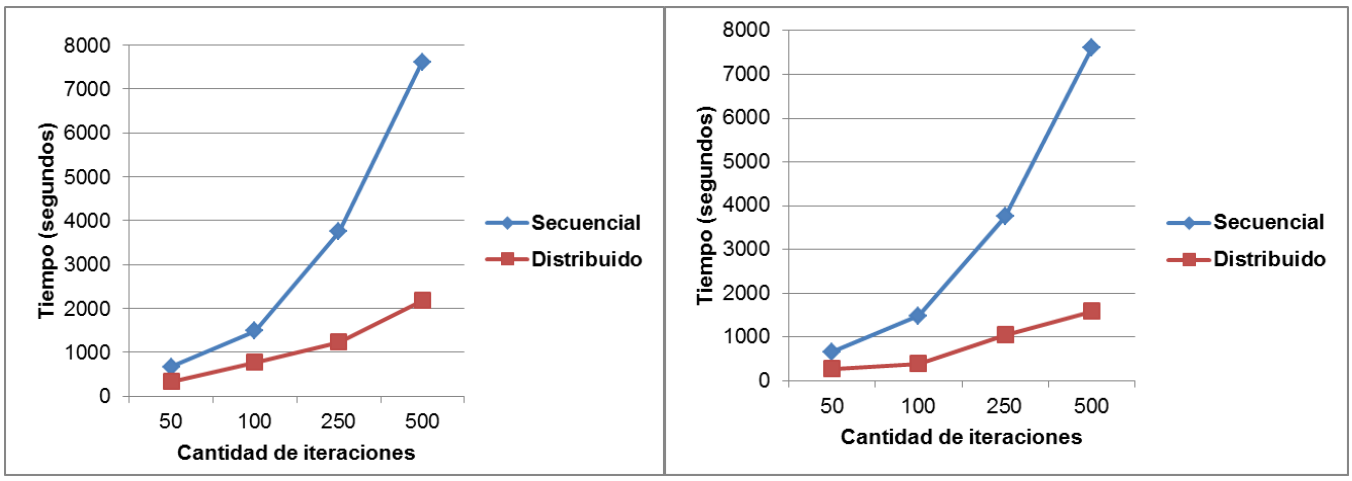
Los experimentos realizados de manera distribuida fueron ejecutados sobre la plataforma de tareas distribuidas T-arenal, desplegada en el laboratorio 402 del Docente 4 de la UCI. Las PCs utilizadas poseen un procesador Intel (R) Pentium (R) 4 a 3.00 GHz y memoria RAM de 1 GB. Todas las pruebas fueron realizadas con misma configuración utilizada durante la experimentación sobre los algoritmos secuenciales. La forma de implementación utilizada para ejecutar los algoritmos AS y MMAS, de manera distribuida, fue del tipo Hormigas Paralelas (ver epígrafe 1.7.1)



(a) Tiempo de ejecución (promedio) (b) Tiempo de ejecución (mejor)

Figura 19 Comparación de los tiempos de ejecución secuencial y distribuido del algoritmo AS.

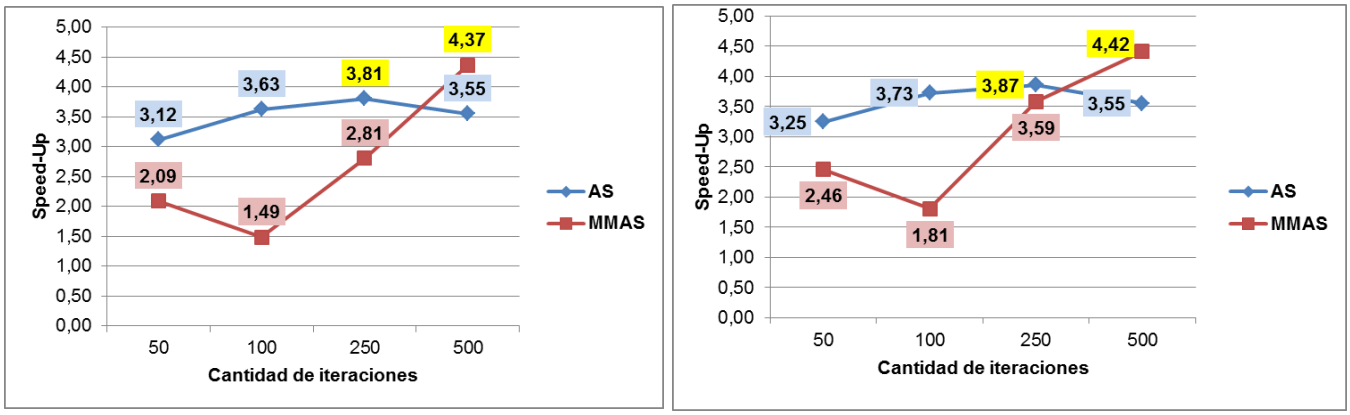
En la Figura 19 se muestra la comparación de los tiempos de ejecución de las versiones secuenciales y distribuidas (usando hasta 10 clientes) del algoritmo AS. Como puede observarse en la Figura 19(a) el promedio de las pruebas realizadas de forma distribuida se ejecutaron en un menor tiempo que las realizadas de forma secuencial. También se puede apreciar, que a medida que la cantidad de trabajo aumenta, el tiempo de ejecución secuencial es mucho mayor que el tiempo de ejecución distribuido. La Figura 19(b) se muestran los resultados de las comparaciones de las pruebas de forma secuencial, con el mejor valor obtenido de las corridas realizadas de forma distribuida en las diferentes iteraciones, apreciándose una disminución del tiempo de ejecución de las pruebas en forma distribuida. Por ejemplo con 500 iteraciones la versión secuencial del algoritmo se demoró más de 3 horas para realizar su ejecución, no obstante la versión distribuida demoró solamente 42 minutos y 2 segundos



(a) Tiempo de ejecución (promedio) (b) Tiempo de ejecución (mejor)

Figura 20 Comparación de los tiempos de ejecución secuencial y distribuido del algoritmo MMAS.

En la Figura 20 se muestran los mismos tipos de comparaciones que se reflejaron en la Figura 19 pero en este caso para el algoritmo MMAS, observándose el mismo comportamiento.

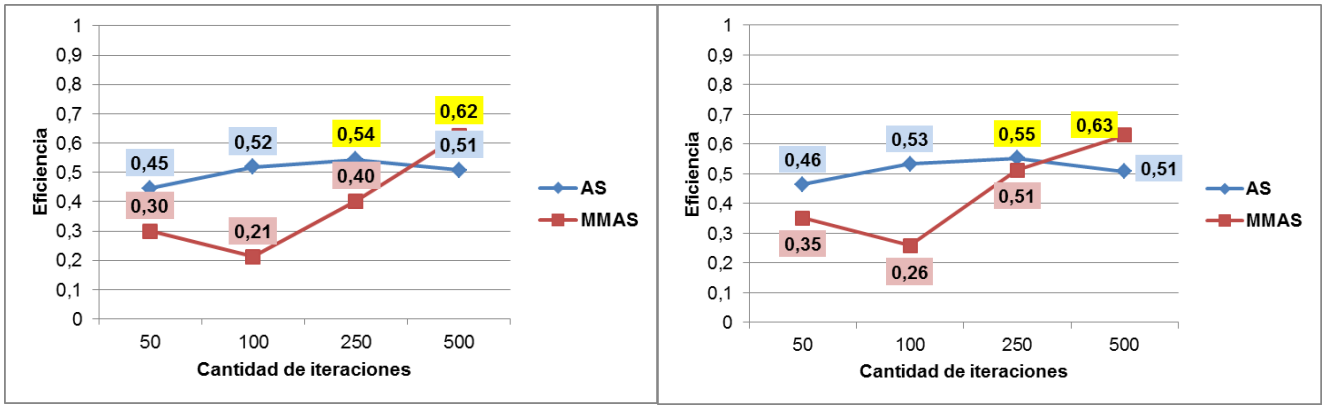


(a) Speed-Up promedio. (b) Speed-Up máximo.

Figura 21 Speed-Up de los algoritmos en función de la cantidad de iteraciones utilizando 7 clientes.

El análisis de los resultados obtenidos en la Figura 21, muestra las ganancias de velocidad que se alcanzaron en las pruebas efectuadas para los algoritmos AS y MMAS, en función de la cantidad de iteraciones utilizadas. En la Figura 21(a) se muestra la ganancia de velocidad promedio lograda en los experimentos, se muestra que la ganancia de velocidad del algoritmo AS siempre estuvo por encima de 3,12 y el algoritmo MMAS después de la iteración 100 alcanza un incremento notable de la misma.

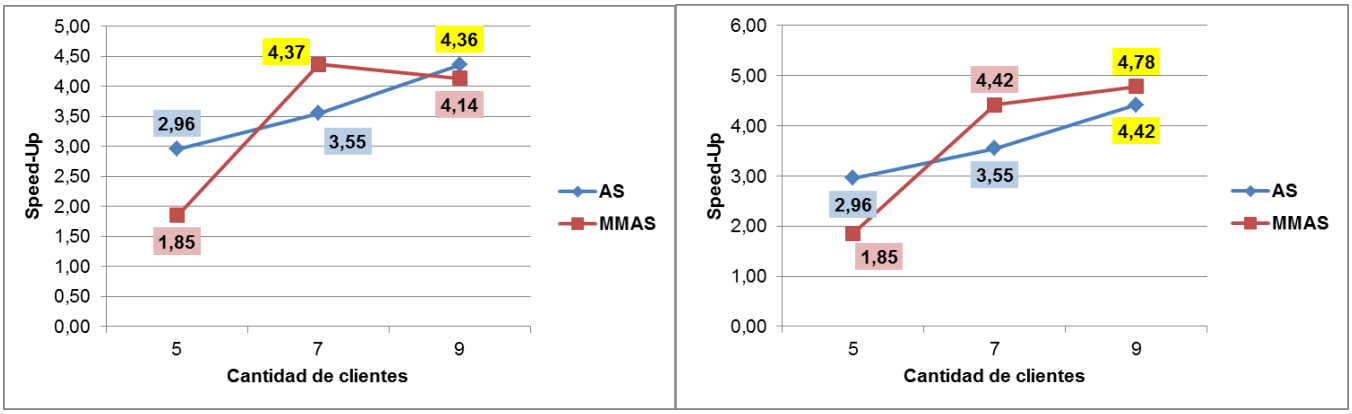
La Figura 21(b) muestra la ganancia de velocidad máxima obtenidas por los algoritmos utilizados, lográndose una reducción del tiempo máxima para AS de 3.87 veces (de 1 hora 42 minutos a 22 minutos y 13 segundos) y para MMAS de 4.42 (de 2 horas y 11 minutos a 29 minutos y 15 segundos).



(a) Eficiencia promedio (b) Eficiencia máxima

Figura 22 Eficiencia de los algoritmos en función de la cantidad de iteraciones utilizando 7 clientes.

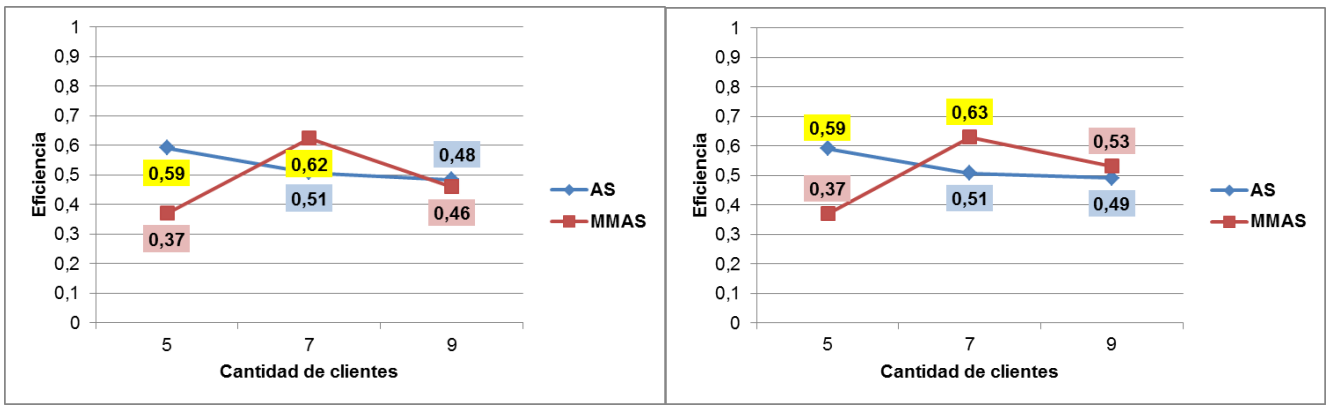
La Figura 22 muestra los la eficiencia lograda por los algoritmos AS y MMAS en función de la cantidad de iteraciones. En la Figura 22(a) y Figura 22(b) se aprecia un análisis de la eficiencia promedio y máxima respectivamente, alcanzadas en las pruebas. Se observa que AS logra una eficiencia máxima de 0.55 y MMAS un 0.63, lo que significa que al aumentar la cantidad de trabajo MMAS tuvo un mejor aprovechamiento de las PCs comparado con AS.



(a) Speed-Up promedio. (b) Speed-Up máximo.

Figura 23 Speed-Up de los algoritmos en función de la cantidad de clientes en 500 iteraciones.

El análisis de la Figura 23 muestra la ganancia de velocidad lograda por los algoritmos AS y MMAS en las pruebas, en función de la cantidad de clientes con un número de iteraciones fija (250). En la Figura 23(a) y Figura 23(b) se evidencia la ganancia de velocidad promedio y máxima de los algoritmos respectivamente, se aprecia un aumento de la ganancia de velocidad en el algoritmo AS a medida que se incrementa el número de clientes, sin embargo este compartimiento se aprecia en MMAS solamente en la Figura 23(b). Con la ganancia de velocidad máxima adquirida por el algoritmo AS, se logró reducir el tiempo 4.42 veces (de 3 horas a 41 minutos y 2 segundos) y MMAS disminuyó el tiempo 4.78 veces (de 2 horas y 11 min a 26 minutos y 55 segundos).



(a) Eficiencia promedio (b) Eficiencia máxima

Figura 24 Eficiencia de los algoritmos en función de la cantidad de clientes en 500 iteraciones.

La Figura 24 la eficiencia lograda por los algoritmos AS y MMAS en función de la cantidad de clientes utilizados con un número fijo de iteraciones (250). La Figura 24(a) muestra la eficiencia promedio y la Figura 24(b) la eficiencia máxima obtenida. La eficiencia máxima obtenida por los algoritmos AS y MMAS es de 0.59 y 0.63 respectivamente, lo que significa que los clientes utilizados estuvieron dedicados más de la mitad del tiempo en la ejecución de los algoritmos de la biblioteca.

Comparación de valores de eficiencia alcanzados sobre la plataforma T-arenal

Para tener una mejor idea de cuán eficiente trabaja la Biblioteca ACO sobre la plataforma T-arenal, se realizó una comparación entre los valores de eficiencia alcanzados por los algoritmos AS y MMAS de forma distribuida, con otros trabajos (ver Tabla 1) en los cuales se ha utilizado la plataforma.

Tabla 1 Trabajos que han utilizado la plataforma T-arenal.

Identificador	Autor(es)	Referencia

Aguilera	MsC. Longendri Aguilera Mendoza	(3)
Trinchet	MsC. Dannier Trinchet Almaguer	(4)

Los resultados mostrados en la Figura 25 destacan que el promedio de eficiencia alcanzado por AS y MMAS estuvieron por encima que los otros trabajos. También se aprecia que los resultados máximos de eficiencia obtenidos por los algoritmos de la Biblioteca ACO solo fueron superamos por el valor alcanzado en el trabajo de Aguilera, considerándose así que la biblioteca tuvo un buen aprovechamiento de los recursos computaciones que emplea para su funcionamiento la plataforma de Tareas Distribuidas T-arenal.

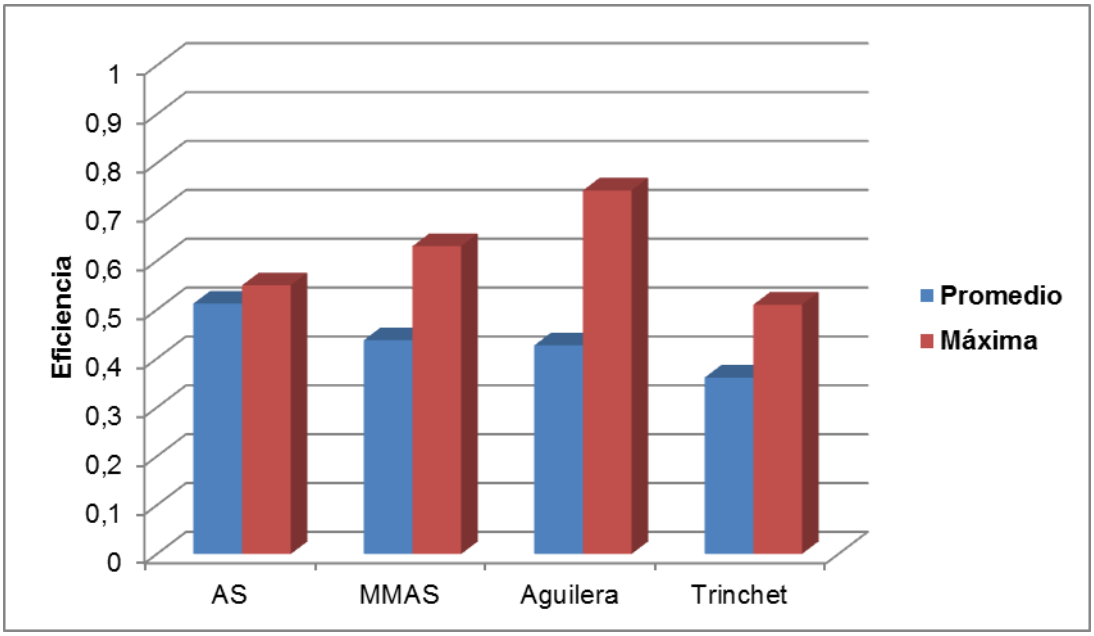


Figura 25 Comparación de eficiencias entre distintos trabajos.

3.6 Valoraciones finales

Como hipótesis de la presente investigación se planteó que mediante el empleo de algoritmos de Optimización basados en Colonias de Hormigas sobre la plataforma de Tareas Distribuidas T-Arenal para resolver problemas de optimización, costosos computacionalmente, aprovechando la capacidad de cómputo de la UCI, es posible disminuir considerablemente el tiempo de ejecución de los mismos. Como variable se identificó el tiempo de ejecución.

Los resultados prácticos que se obtuvieron de la modelación y resolución del QAP, probaron la hipótesis antes mencionada, demostrando que la versión distribuida de la Biblioteca ACO, utilizando el

modelo maestro-esclavo, se ejecutó en un tiempo mucho menor que su versión secuencial, evidenciándose que en correspondencia con el aumento de la cantidad de trabajo, la versión distribuida del algoritmo se ejecutaba con mayor rapidez que la versión secuencial.

3.7 Conclusiones

En este capítulo se demostró la eficiencia de la biblioteca y su desempeño exitoso en el caso de estudio propuesto. Se aplicó la biblioteca al problema de optimización combinatoria QAP, costoso computacionalmente, de forma secuencial y distribuida, obteniéndose una reducción significativa de tiempo.

CONCLUSIONES GENERALES

Como resultado principal de la investigación se obtuvo una biblioteca implementada en Java con tres algoritmos de Optimización basados en Colonias de Hormigas. La biblioteca proporciona funcionalidades que brindan facilidades para la modelación y solución de problemas de optimización que requieren un amplio poder de cómputo.

Mediante la resolución del caso de estudio, se probó que la implementación de la biblioteca para la aplicación de algoritmos ACO sobre la plataforma T-arenal en la resolución de problemas de optimización, costosos computacionalmente, de forma distribuida, disminuye el tiempo de solución de los mismos.

RECOMENDACIONES

Para futuras mejoras de esta solución se recomienda:

- Implementar una o varias variantes de técnicas de Búsqueda Local para integrarlas la biblioteca.
- Implementar otros algoritmos basados en ACO para así ampliar las posibilidades de aplicación.
- Probar la Biblioteca ACO en otros problemas de optimización.

BiBLIOGRAFÍA

1. Stützle, Thomas y Dorigo, Marco. *Ant Colony Optimization.* s.l. : A Bradford Book, 2004.

2. Dorigo, Marco y Di Caro, G. *The Ant Colony Optimization meta-heuristic.* [ed.] Marco Dorigo, G Di Caro y F Glover. 1999.

3. Aguilera Mendoza, Longendri. *Sistema de cómputo distribuido aplicado a la Bioinformática.* Universidad de las Ciencias Informáticas. 2008. Tesis.

4. Trinchet Almaguer, Dannier. *Algoritmo paralelo para el modelado de yacimientos lateríticos.* La Habana : s.n., 2010. Tesis.

5. Alba, Enrique, [ed.]. *Parallel metaheuristics : a new class of algorithms.* Hoboken : John Wiley & Sons, Inc.,, 2005.

6. Talbi, El-Ghazali. *METAHEURISTICS FROM DESIGN TO IMPLEMENTATION.* Hoboken : John Wiley & Sons, Inc., 2009.

7. Papadimitriou, C.H y Steiglitz, K. *Combinatorial optimization: algorithms and complexity.* s.l. : Prentice-Hall, Inc., 1982.

8. *Metaheuristics in combinatorial optimization: Overview and conceptual comparison.* Blum, C. y Roli, A. s.l. : ACM Computing Surveys, 2003, ACM Computing Surveys.

9. *The Hyper-Cube Framework for Ant Colony Optimization.* Blum, C. y Dorigo, M. 2004, IEEE Transactions on Systems, Man, and Cybernetics.

10. Papadimitriou, C.H y Steiglitz, K. *Combinatorial optimization: algorithms and complexity.* s.l. : Prentice-Hall, Inc., 1982.

11. Brassard, G y Bratley, P. *Fundamentals of algorithmics. .* s.l. : Prentice-Hall, Inc., 1996.

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

13. Kirkpatrick, S, Gelatt, C.D y Vecchi, M.P. *Optimization by simulated annealing.* s.l. : Science, 1983.

14. Cerny, V. A thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications.* 1985.

15. Higuera Cabañes, Clara. *DISEÑO DE UN MÉTODO SELECTIVO INSPIRADO EN ENFRIAMIENTO SIMULADO APLICADO A UN PROCESO BIOQUÍMICO.* Universidad Complutense. Madrid : s.n., 2010.

16. Aarts, E. H. L. y Lenstra, J. K. *Local Search in Combinatorial Optimization.* s.l. : Wiley, 1997.

17. *Tabu search.* Glover, F. y Laguna, M. s.l. : Kluwer, 1997.

18. *Uso de búsqueda tabú en la solución del problema de asignación cuadrática.* Quevedo Orozco, Dagoberto Ramón y Ríos Mercado, Roger Z. . No. 48, 2010, Vol. XIII.

19. Salgado Reyes, Rigoberto Leander. *Librería paralela de algoritmos evolutivos aplicados a problemas computacionalmente costosos.* Universidad de las Ciencias Infotmáticas. Ciudad Habana : s.n., 2009. Tesis.

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

21. *Particle Swarm Optimization, Methods, Taxonomy and Applications.* Sedighizadeh, Davoud y Masehian, Ellips. 5, 2009, Vol. 1. 1793-8201.

22. No Free Lunch Theorems. [En línea] [Citado el: 3 de 05 de 2012.] http://www.no-free-lunch.org/.

23. Grasse, P. *La Reconstruction du nid et les Coordinations Inter-Individuelles chez Bellicositerm.es Natalensis et Cubitermes sp. La theorie de la Stigmergie: Essai d'interpretation du Comportement des Termites Constructeurs.* s.l. : Insect.

24. —. *Termitologia, Tome II.* Paris : Fondation des Societes, 1984.

25. Dorigo, Marco. *Optimization, learning and natural algorithms.* Milano : Italy, 1992. Tesis.

26. Dorigo, Marco y Stützle, Thomas. *The ant colony optimization metaheuristic: Algorithms, applications and advances.* [ed.] F Glover y G Kochenberger. s.l. : Handbook of Metaheuristics, 2003.

27. Pedemonte, Martín . *Ant Colony Optimization para la resolución del Problema de Steiner Generalizado.* 2009. Tesis.

28. Corne, D, Dorigo, Marco y Glover, F. *New Ideas in Optimization.* s.l. : McGraw-Hill, 1999.

29. Dorigo, Marco y Gambardella, L. *Ant colony system: A cooperative learning approach to the traveling salesman problem.* s.l. : IEEE Transactions on Evolutionary Computation, 1997.

30. Maniezzo, V. *Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem.* s.l. : INFORMS Journal on Computing, 1999.

31. Merkle, D y Middendorf, M. *Swarm Intelligence.* s.l. : In Search Methodologies Springer, 2005.

32. *Ant system: Optimization by a colony of cooperating agents.* Dorigo, Marco, Maniezzo, V y Colorni, A. 1996, IEEE Transactions on Systems, Man, and Cybernetics-Part B.

33. Dorigo, Marco, Maniezzo, V y Colorn, A. *Positive feedback as a search strategy.* Politecnico di Milano. 1991. Technical report.

34. Stützle, Thomas y Hoos, H. *Improving the Ant System: A detailed report on the MAX-MIN Ant System.* 1996. Technical report.

35. *MAX-MIN ant system.* Stützle, Thomas y Hoos, H. 2000, Future Generation Computer Systems.

36. Stützle, Thomas y Hoos, H. *The MAX-MIN Ant System and local search for combinatorial optimization problems.* 1999.

37. Chirico , Ugo. *A Java Framework for Ant Colony Systems .* Siemens Informatica S.p.A. Roma : s.n. Informe.

38. Hammerl, Thomas. *Ant Colony Optimization for tree and hypertree descompositions.* Vienna University of Technology. 2009. Tesis.

39. Rauber, Thomas. *Parallel Programming: for Multicore and Cluster Systems.* s.l. : Springer, 2010. ISBN-10: 364204817X.

40. G Couloris JD, Kinberg T. *Distributed Systems - Concepts and Design.* Vol. 4th Edition.

41. A, Tanenbaum y MV., Steen. *Distributed Systems: Principles and Paradigms.* USA : Prentice Hall, Pearson Education, 2002.

42. Tanenbaum, A. *Distributed Operating Systems.* s.l. : Prentice Hall, 1995.

43. Puder, A, ROMER, K y PILHOFER, F. *Distributed Systems Architecture.* s.l. : Morgan. Vol. A Middleware Approach.

44. Quintero Henríquez, Adrián. *Algoritmos paralelos de optimización por Nube de Partículas.* Universidad de las Ciencias Informáticas. Ciudad Habana : s.n., 2010. Tesis.

45. Miki, M., Hiroyasu, T. y Negami, M. *Distributed Genetic Algorithms with Randomized Migration Rate.* 1999. págs. 689-694. in IEEE Proceedings of Systems, Man and Cybernetics Conference SMC'99.

46. Luque Polo, Gabriel Jesús. *Resolución de Problemas Combinatorios con Aplicación Real en Sistemas Distribuidos.* Lenguajes y Ciencias de la Computación, UNIVERSIDAD DE MÁLAGA. 2006. TESIS DOCTORAL.

47. *A Parallel Implementation of Ant Colony Optimization.* Randall , M. y Lewis, A. 2002, Journal of Parallel and Distributed Computing,.

48. [En línea] [Citado el: 17 de mayo de 2012.] http://www.scribd.com/doc/3062020/Capitulo-I-HERRAMIENTAS-CASE.

49. UML, BPMN and Database Tool for Software Development. [En línea] [Citado el: 15 de mayo de 2012.] http://www.visual-paradigm.com.

50. Marañón, Gonzalo Álvarez. Gonzalo Álvarez Marañón. *Java.* [En línea] CSIC. Todos los derechos reservados, 1997-1999. [Citado el: 15 de 1 de 2012.] http://www.iec.csic.es/criptonomicon/java/quesjava.html.

51. García Jacas, Cesar Raúl. *Front-end de la Plataforma de Tareas Distribuidas. Interfaz de Escritorio T-arenal Versión 2.0.* Manual del usuario.

52. Koopmans, Tjalling C. y Beckmann, Martin. *Assignment problems and the location of economic activities.* s.l. : Econometrica, 1957.

53. Stützle, Thomas. *MAX-MIN Ant System for Quadratic Assignment Problems.* s.l. : Technische Hochschule Darmstardt, 1997.

54. Cela, Eranda. *The Quadratic Assignment Problem Theory and Algorithms.* s.l. : Kluwer Academic Publishers, 1998.

55. Garey, M. R. y Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco : W. H. Freeman, 1979.

56. Steinberg, L. *The backboard wiring problem: A placement algorithm.* s.l. : SIAM Review, 1961.

57. Dickey , J. W. y Hopkins, J. W. *Campus building arrangement using topaz.* s.l. : Transportation Science, 1972.

58. Applegate, D. L., y otros, y otros. *The Traveling Salesman Problem.* 2006.

59. Albers, S y Mitzenmacher, M. *Average-Case Analyses of First Fit and Random Fit Bin Packing.* 2000.

60. *Simplified Ant Colony System applied to the Quadratic Assignment Problem.* Barton, Alan J. s.l. : NRC Publications Record / Notice d'Archives des publications de CNRC, 2005.

61. [En línea] [Citado el: 25 de mayo de 2012.] http://www.opt.math.tu-graz.ac.at/qaplib/inst.html.

62. Arito, Franco Luis Alejandro. *Algoritmos de Optimización basados en Colonias de Hormigas aplicados al Problema de Asignación Hormigas aplicados al Problema de Asignacióon.* Universidad Nacional de San Luis, Argentina. 2010. Tesis.

63. *A simplex method for function minimization.* Nelder, J. A. y Mead, R. 1964, The Computer Journal.

64. Wu, J. *Distributed System Design.* s.l. : CRC-Press., 1998.

65. *Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison.* Blum, C. y Roli, A. 35, ACM Computing Surveys, Vol. 3.

66. Palomar, Isabel. [En línea] 7 de Noviembre de 2010. [Citado el: 10 de Diciembre de 2011.] http://genetic-algorithms1.blogspot.com/search/label/ALGORITMOS%20GENETICOS..

67. *Algoritmo basado en la optimización mediante colonias de hormigas para la resolución del problema del transporte de carga desde varios orígenes a varios destinos.* Barcos, Lucía, Rodríguez, Victoria M y Álvarez, Mª Jesús. 2002.

68. Ostf eld, Avi, [ed.]. *ANT COLONY OPTIMIZATION-METHODS AND APPLICATIONS.* s.l. : InTech.

69. Palomar, Isabel. Look Towards A New Future. *Blog (Isabel Palomar).* [En línea] 11 de 7 de 2010. [Citado el: 20 de 9 de 2011.] http://genetic-algorithms1.blogspot.com/search/label/ALGORITMOS%20GENETICOS.

70. Tsutsui , S. y Liu, L. *Cunning Ant System for Quadratic Assignment Problem with Local Search and Parallelization.* Missouri Estimation of Distribution Algorithms Laboratory, University of Missouri. St. Louis : s.n., 2007.

71. *A Parallel Ant Colony Algorithm for Bus Network Optimization.* Yang, Z., Yu, B. y Cheng, C. 2007, Computer-Aided Civil and Infrastructure Engineering,.

72. *Exchange Strategies for Multiple Ant Colony System.* Ellabib, I., Calamai, P.H. y Basir, O. A. 2007, Information Sciences.

ANEXOS

Anexo 1: Fichero de configuración de los parámetros de la biblioteca ACO.

parameters.properties

```
#Data
#ACO
alpha=1
beta=1
eRate=0.1
tIterations=100
nAnt=100

#Para evaluar las métricas(evaluarlas por iteraciones evaIteration=true, en caso contrario  evaIteration=false)
evaIteration=true

#Para el algoritmo ACS
q0=0.3
eRate0=0.2

#Para el algoritmo MMAS
#Porcentaje de la cantidad de iteraciones para reinicializar la matriz de feromona
resetPercent=0

#Para actualizar la matriz de feromona utilizando(uPBestGlobal=true si se quiere actualizar
#con el mejor valor global, uPBestGlobal=false si se quiere actualizar con el mejor valor de la iteracion)
uPBestGlobal=false
```

GLOSARIO DE TÉRMINOS

AS Del Inglés *Ant System*, Sistema de Hormigas

ACO Del Inglés *Ant Colony Optimization*, Optimización basada en Colonias de Hormigas

ACS Del Inglés *Ant Colony System*, Sistema de Colonias de Hormigas

PSO Del Inglés *Particle Swarm Optimization*, Optimización por Enjambre de Partículas.

EA Del Inglés *Evolutionary Algorithms*, Algoritmos Evolutivos

TS Del Inglés Search Tabu, Búsqueda Tabú

UCI Universidad de las Ciencias Informáticas

IDE Del Inglés *Integrated Development Enviroment*, Entorno Integrado de Desarrollo

MMAS Del Inglés *MAX-MIN Ant System*, MAX-MIN Sistema de Hormigas

QAP Del Inglés *Quadratic Assignment Problem,* Problema de Asignación Cuadrática