Universidad de las Ciencias Informáticas
Ciudad de La Habana
Facultad 6

# TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

## Título: Biblioteca para la optimización basada en Algoritmos Genéticos distribuidos sobre la plataforma T-Arenal.

Autor: Ernesto Luis García Pinal.
Tutor: Dannier Trinchet Almaguer.

*La Habana, junio de 2012*
*"Año 54 de la Revolución"*

# DECLARACIÓN DE AUTORÍA 2012

Declaro que soy el único autor del presente trabajo y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

"[Insertar nombre(s) de autor(es)]"
**__Ernesto Luis García Pinal____**

"[Insertar nombre(s) de tutor(es)]"
**___DannierTrinchet Almaguer____**

**Síntesis del Tutor:**

**Especialidad de graduación:** MsC. Lic. Dannier Trinchet Almaguer

**Categoría docente:** Profesor Asistente y Máster en Ciencias de la Computación.

**Grado científico:** Máster

**Título de la especialidad de graduado:** Licenciado en Ciencias de la Computación

**Años de graduado:** Tiene 6 años de experiencia en líneas de investigación relacionadas con la Inteligencia Artificial y la Computación Paralela y Distribuida.

**Institución donde se graduó:** Universidad de Oriente

Cuenta con varias participaciones (publicaciones) en eventos (revistas) internacionales y nacionales.

**Datos de Contacto:**

MsC. Lic. Dannier Trinchet Almaguer.

Universidad de las Ciencias Informáticas, Habana, Cuba.

Correo electrónico: trinchet@uci.cu

*A mi familia por su apoyo, dedicación, ejemplo…*
*Todos mis logros son para ustedes.*

*A mis padres y mis hermanos que han sido un ejemplo a seguir y han hecho posible todos mis logros.*

*A mis maestros y profesores de todas las enseñanzas cursadas que, junto con mi familia, fueron el eslabón fundamental en mi formación como estudiante y como profesional.*

*A mi tutor Trinchet por su ayuda en el desarrollo de este trabajo.*

*A los Dr.Cs Juan Julian Merelo y Maribel García Arenas.*

*A mi novia y mis amigos por compartir conmigo estos años y por su apoyo en todo momento.*

*A todos aquellos que, de una forma u otra, tuvieron que ver con la realización de este trabajo y me brindaron su ayuda cuando la necesité.*

*Gracias.*

A diario se enfrentan problemas de optimización muy complejos y prácticamente imposibles de resolver en una sola unidad de procesamiento, por su alto costo computacional. La UCI cuenta con un gran poder de cómputo y con una plataforma denominada T-Arenal, que aprovecha dichos recursos para resolver tareas de forma distribuida.

Los Algoritmos Genéticos (AG), una de las líneas más prometedoras de la inteligencia artificial, son una técnica metaheurística inspirada en la evolución biológica y su base genético-molecular. Mientras el poder de la evolución gana un reconocimiento cada vez más generalizado, los Algoritmos Genéticos se utilizan para abordar una amplia variedad de problemas en un conjunto de campos sumamente diversos demostrando claramente su capacidad y potencial.

El presente trabajo se enfoca en el estudio de este método y en el desarrollo de una biblioteca eficiente y de propósito general basada en Algoritmos Genéticos que permite resolver problemas de optimización de alto costo computacional de forma secuencial y distribuida sobre la Plataforma de Tareas Distribuidas T-arenal. La biblioteca se aplicó de forma eficiente a problemas de asignación cuadrática (QAP), obteniéndose buenos resultados y demostrando la gran capacidad de los algoritmos evolutivos y en especial de los Algoritmos Genéticos para resolver problemas de optimización complejos y computacionalmente costosos.

**Palabras clave:** computación distribuida, Algoritmos Genéticos, programación evolutiva, optimización, metaheurística.

***OPTMIZATION LIBRARY BASED IN GENETIC ALGORITHMS DISTRIBUTED ON T-ARENAL PLATFORM***

*Problems of optimization are faced at present times which are very complex and virtually impossible to solve a single processing unit for its high computational cost. The University of Informatics Sciences (UIS) has a great computing power and a platform called T-Arenal, which uses these resources to solve tasks in a distributed manner.*

*Genetic algorithms (GA), one of the most promising lines of artificial intelligence, are a metaheuristic technique inspired by biological evolution and molecular-genetic basis. While the power of evolution wins increasingly widespread recognition, genetic algorithms are used to address a wide variety of problems in a highly diverse set of fields demonstrating their ability and potential.*

*In this paper we developed a library of genetic algorithms, efficient, general-purpose, and capable of solving optimization problems, computationally expensive, sequentially and in a distributed manner on the platform T-Arenal. The library efficiently applied to quadratic assignment problems (QAP), obtaining very good results and demonstrating grate capacity of the evolutionary algorithms and especially genetic algorithms to solve complex and computationally expensive optimization problems.*

***Keywords:*** *distributed computation, genetic algorithms, evolutionary programming, optimization, metaheuristic.*

## Introducción

En un mundo cada vez más complejo y competitivo, la toma de decisiones debe ser abordada de una manera racional y óptima. A diario se enfrentan problemas donde se tienen varias soluciones y se debe determinar cuál de ellas tomar, como por ejemplo, qué camino seguir para ir de la casa al trabajo, los que constituyen problemas sencillos de optimización. De forma genérica, puede definirse la optimización como la ciencia encargada de determinar las mejores soluciones a problemas matemáticos que a menudo modelan una realidad física.

Con el amplio desarrollo de la computación y la informática se han desarrollado diversas técnicas para tratar de resolver dicho tipo de problemas. Las técnicas exactas garantizan encontrar la solución óptima para cualquier instancia de cualquier problema en un tiempo acotado. Sin embargo, existen problemas que por su complejidad no pueden ser abordados con éxito por métodos exactos; de ahí el surgimiento de métodos aproximados para tratar de resolverlos. Dentro de estos métodos se encuentran las denominadas técnicas metaheurísticas. A diferencia de los métodos exactos, las metaheurísticas permiten hacer frente a instancias de problemas de gran tamaño, entregando soluciones satisfactorias en un plazo razonable. Aunque no garantizan encontrar las soluciones óptimas globales han recibido gran popularidad en los últimos 20 años.

En los años 1970, de la mano de John Henry Holland, surgió una de las líneas más prometedoras de la inteligencia artificial, la de los **Algoritmos Genéticos (AG)**. Los AG constituyen una técnica metaheurística inspirada en la evolución biológica y su base genético-molecular. Dichos algoritmos hacen evolucionar a 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. Son incluidos dentro de los algoritmos evolutivos, que incluyen también las estrategias evolutivas, la programación evolutiva y la programación genética.

En la actualidad las aplicaciones de dichos algoritmos están enfocadas a resolver problemas de optimización. Mientras el poder de la evolución gana un reconocimiento cada vez más generalizado, los Algoritmos Genéticos se utilizan para abordar una amplia variedad de problemas en un conjunto de campos sumamente diversos como la física, la química, la ciencia de los materiales, la biología, la

biotecnología, la farmacología, los videos juegos y entrenadores virtuales demostrando claramente su capacidad y potencial **(Palomar, 2010)**.

Con el desarrollo de la humanidad han aparecido problemas cada vez más complejos, donde obtener soluciones óptimas requiere un alto costo computacional y de tiempo. A pesar de la calidad de las metaheurísticas para resolver problemas de gran tamaño, muchas veces no basta con una sola unidad de procesamiento debido al alto costo computacional. Una alternativa entonces es la utilización de un Sistema Distribuido (SD).

Un SD es un sistema de procesamiento de información que está compuesto por varias computadoras independientes que cooperan entre ellas mediante una red de comunicaciones con el fin de alcanzar un objetivo específico **(A. Puder, 2006)**. La computación distribuida brinda acceso transparente al poder de cómputo y almacenamiento de muchas computadoras independientes que un usuario necesita para realizar una determinada tarea, al tiempo que permite alcanzar altos índices de rendimiento y confiabilidad mediante el aprovechamiento de esos recursos computacionales.

La Universidad de Las Ciencias Informáticas (UCI), es un centro con un alto desarrollo tecnológico que cuenta con gran cantidad de unidades de procesamiento. La línea de Bioinformática del Centro de Tecnologías de Gestión de Datos (DATEC), perteneciente a la Facultad 6 de dicha universidad, cuenta con un proyecto denominado Plataforma de Servicios Bioinformáticos. Entre los módulos que lo componen se encuentra la Plataforma de Cálculo Distribuido T-Arenal cuyo objetivo es resolver problemas computacionalmente costosos de forma distribuida, utilizando el gran poder de cómputo que posee la universidad.

T-Arenal es un sistema distribuido de propósito general que permite resolver problemas que requieren de altas prestaciones de cómputo, distribuyendo la carga de trabajo entre los diferentes clientes disponibles **(Jacas)**. En estos momentos la Plataforma se encuentra desplegada en la universidad y se ha utilizado para resolver problemas complejos: docking molecular **(Aguilera Mendoza, 2008)** y modelado de yacimientos lateríticos **(Trinchet Almaguer, 2010)**, obteniéndose buenos resultados. Sin embargo, ésta no cuenta con las herramientas necesarias para dar solución eficiente a problemas de optimización, resultando un tanto compleja la modelación y resolución de un problema de este tipo, debido a que es

necesario implementar totalmente el problema y el algoritmo que le dará solución; lo que trae consigo un aumento considerable del tiempo requerido para su resolución.

Por todo lo anteriormente expuesto surge como **Problema científico** para este trabajo de diploma:
¿Cómo resolver problemas de optimización, computacionalmente costosos, aprovechando los recursos de cómputo de la universidad?

Definiéndose como **Objeto de estudio** los Algoritmos Genéticos distribuidos, enmarcando como **Campo de acción** los Algoritmos Genéticos distribuidos en T-Arenal.
Para darle solución al problema científico anteriormente enunciado surge como **Objetivo general** de la investigación:
Desarrollar una biblioteca para resolver problemas de optimización computacionalmente costosos mediante Algoritmos Genéticos distribuidos en T-Arenal.

Para lograr los objetivos trazados se acometen las siguientes **Tareas de la investigación:**

1. Revisión bibliográfica de los Algoritmos Genéticos.
2. Diseño de la biblioteca.
3. Implementación de la versión secuencial de la biblioteca.
4. Revisión y estudio de los esquemas distribuidos del algoritmo.
5. Implementación de la versión distribuida de la biblioteca.
6. Modelado y resolución de un *Assigment Problem*[1].
7. Realización de pruebas de la aplicación de la biblioteca en la plataforma de tareas distribuidas T-Arenal.

### Hipótesis investigativa.

Si se emplean Algoritmos Genéticos sobre la plataforma de cálculo distribuido T-Arenal para resolver problemas de optimización computacionalmente costosos, según las posibilidades de cómputo de la UCI, es posible disminuir considerablemente el tiempo requerido y aumentar la calidad de la solución.

### Variables:

[1] Problema de asignación.

- ✓ Tiempo de ejecución
- ✓ Calidad de la solución

En la investigación se hizo necesario aplicar algunos métodos científicos que tienen su sustento en la concepción materialista dialéctica y facilitan la recopilación de la información necesaria para el desarrollo de una biblioteca donde se puedan resolver problemas de optimización computacionalmente costosos mediante Algoritmos Genéticos sobre la plataforma T-Arenal.

De los teóricos se utilizarán el Análisis Histórico – Lógico, Analítico – Sintético y la Modelación

- El método Histórico – Lógico permite determinar las características y conceptos fundamentales de los Algoritmos Genéticos y la programación distribuida para realizar el estudio de la trayectoria real de determinados elementos que servirán de guía en el desarrollo de una biblioteca para resolver problemas de optimización computacionalmente costosos mediante Algoritmos Genéticos sobre la plataforma T-Arenal.
- El método Analítico – Sintético permite realizar un análisis de la documentación de cada una de las herramientas que se utilizarán en el sistema a desarrollar.
- El método Modelación se utiliza en la modelación de diagramas.

De los métodos empíricos se utilizó:

- El experimento, permitiendo realizar pruebas experimentales a la biblioteca una vez que fue desarrollada.

La estructura del documento de la presente investigación se define de la siguiente forma**:**

**El Capítulo 1: Fundamento Teórico.** Se realiza un estudio de algunos conceptos y elementos teóricos relacionados con los Algoritmos Genéticos, así como de las herramientas y tecnologías que se utilizarán para el desarrollo del presente trabajo de diploma.

**El Capítulo 2: Diseño e Implementación.** Se lleva a cabo el diseño del sistema y cómo será implementada la biblioteca de forma distribuida para la plataforma T-Arenal.

**El Capítulo 3: Resultados y Discusión.** Se exponen los resultados de la investigación mediante el modelado de un problema de asignación cuadrática y la comparación de resultados de la ejecución secuencial y la distribuida.

# CAPÍTULO 1: Fundamento Teórico.

## 1.1. Introducción

En este capítulo se abarcan temas de la investigación que están relacionados con algunos conceptos básicos y elementos teóricos sobre los Algoritmos Genéticos para lograr un dominio del problema que se plantea. Se realiza un estudio de algunos sistemas existentes que ayudan a resolver problemas de optimización computacionalmente costosos mediante Algoritmos Genéticos, teniendo en cuenta las características de la plataforma T-Arenal. Además se abordarán temas relacionados con las tecnologías y herramientas que se utilizarán para el desarrollo de la biblioteca que se pretende con el presente trabajo de diploma.

## 1.2. Problemas de Optimización.

Los problemas de optimización sin restricciones consisten en minimizar el valor real de una función $f$ de $N$ variables. Por este medio se puede plantear que para encontrar el mínimo local, sería encontrar un punto $X^*$ tal que:

$$f(x^*) \leq f(x) \text{ para todo } x \text{ cercano } x^*. \quad (1.1)$$

Es estándar expresar el problema de la siguiente forma:

$$\min_x f(x) \quad (1.2)$$

Entendemos por 1.1 como un optimizador local, refiriéndose a $f$ como la función objetivo y a $f(x^*)$ como el mínimo local. Este problema es diferente del problema de encontrar el mínimo global, donde el mismo es encontrado de la siguiente forma:

$$f(x^*) \leq f(x) \text{ para todo } x. \quad (1.3)$$

Algunas veces la maximización tiene más sentido; para maximizar una función, simplemente invertimos el signo de la propia función, por ejemplo, maximizar $1 - x^2$ en el intervalo $-1 \leq x \leq 1$ que es el mismo problema que minimizar $x^2 - 1$ sobre el mismo intervalo **(Reyes, 2009)**.

El problema de optimización con restricciones es minimizar una función $f$ sobre un conjunto $U \subset \Re N$, luego un minimizador local es un $x^* \in U$ tal que:

$$f(x^*) \leq f(x) \text{ para todo } x \in U \text{ cercano } x^*.$$

(1.4)

De forma similar se expresa:

$$\min_x \in U \, f(x^*)$$

(1.5)

El minimizador global es un punto $x^* \in U$ tal que:

$$f(x^*) \leq f(x) \text{ para todo } x \in U \ .$$

(1.6)

### 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" **(Alba, 2005; Blum, y otros, 2003; Glover, y otros, 2002)**.

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.

#### 1.3.1. Clasificación de las metaheurísticas

Dada las características de los algoritmos metaheurísticos son clasificados en dos grupos: algoritmos basados en trayectoria y los algoritmos basados en poblaciones. En la Figura 1 se pueden observar algunas de las técnicas metaheurísticas dentro de estas clasificaciones.

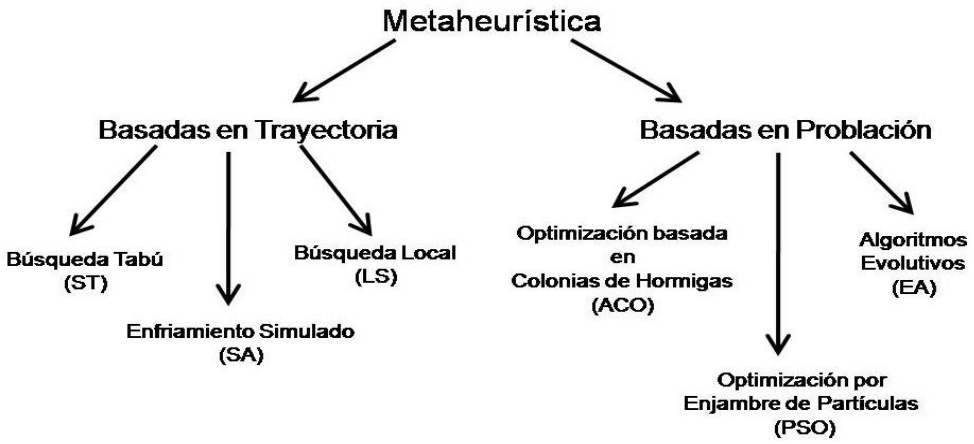


**Figura 1: Clasificación de las metaheurísticas**

La construcción de algoritmos metaheurísticos 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. Con la intensificación se garantiza explorar a fondo las soluciones más prometedoras **(Alba, 2005)**.

### 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. Muchas de las metaheurísticas basadas en poblaciones son bioinspiradas[2].

Algunos de los más populares de estos algoritmos son:

- Algoritmos Evolutivos o *Evolutionary Algorithms* (EA) **(Palomar, Isabel, 2010)**.
- Optimización basada en Colonias de Hormigas o *Ant Colony Optimization* (ACO) **(Blum, y otros, 2003)**.

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

- Optimización por Enjambre de Partículas o *Particle Swarm Optimation* (PSO) **(The particle swarm: Explosion, stability, and convergence in a multidimensional complex space., 2002)**.

Dentro de los Algoritmos Evolutivos se encuentran los Algoritmos Genéticos, los cuales constituyen la técnica metaheurística 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 **(Wolpert, 1996)**. El segundo elemento tiene que ver con la efectividad de la técnica en la resolución de problemas de optimización tanto continuos como discretos, esto permitiría al proyecto productivo enfrentar mayor variedad de problemas complejos de optimización debido a que están en desarrollo otras soluciones basadas en Optimización por Enjambre de Partículas y Optimización por Colonia de Hormigas. Estas propuestas pudieran incluso hibridarse para obtener soluciones de mayor calidad y precisión.

### 1.4. Algoritmos Genéticos

Los Algoritmos Genéticos fueron inventados en 1975 por John Holland, de la Universidad de Michigan. Los AG son algoritmos de optimización, es decir, tratan de encontrar la mejor solución a un problema dado entre un conjunto de soluciones posibles. Los mecanismos de los que se valen los AG para llevar a cabo esa búsqueda pueden verse como una metáfora de los procesos de evolución biológica.

Los AG son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acuerdo con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin en 1859. Por imitación de este proceso, los AG son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores óptimos del problema depende, en buena medida, de una adecuada codificación de las mismas **(Londoño, 2006)**.

En la Teoría de la Evolución (TE), los individuos mejor adaptados al medio donde se encuentran son los que sobreviven y se reproducen, dando lugar a una descendencia que hereda sus mejores características. También se presentan variaciones en los genes por medio de la combinación (reproducción sexual) o por cambios aleatorios (mutaciones), así surgen de forma natural más características que a su vez son puestas a prueba y permiten a sus portadores sobrevivir y reproducirse si son las más adecuadas para el ambiente que les rodea.

Los AG son la analogía a lo que ocurre biológicamente, pero aplicada a encontrar la mejor solución, lo que se conoce como problemas de optimización. Cada solución es un individuo. El método de selección se llama función de ajuste (*fitness*) e indica la aptitud (valor) de cada individuo para resolver el problema. Existen cromosomas con sus genes y sus respectivos alelos. A los cromosomas se les aplican operadores genéticos, equivalentes a la mutación y reproducción sexual, de manera que se generan y ponen a prueba nuevas soluciones. Los AG constituyen un método para probar múltiples soluciones autogeneradas a un problema **(Veloso, y otros, 2011)**.

**Aplicaciones de los Algoritmos Genéticos**

Los Algoritmos Genéticos han sido aplicados con éxito en infinidad de problemas, actuando como un proceso de cómputo que emula la forma de proceder de la evolución genética y opera sobre una población de individuos que representan las soluciones potenciales a un determinado problema. Los Algoritmos Genéticos son herramientas para optimizar casi cualquier problema, ya que éstos pueden trabajar con base en la descripción del medio ambiente, lo cual permite resolver muchos problemas de optimización.

Una de las áreas donde se aplican es en el aprendizaje automático. En inteligencia artificial se conocen como sistemas expertos o programas para la resolución de problemas a los que se cuestiona sobre qué hay que hacer y los cuales dan respuestas. Hay ejemplos de sistemas expertos, por ejemplo, en medicina, abogacía, entre otros **(Palomar, 2010)**.

La aplicación más común de los Algoritmos Genéticos ha sido la solución de problemas de optimización, en donde han mostrado ser muy eficientes y confiables. Sin embargo, no todos los problemas pudieran ser apropiados para la técnica y se recomienda en general tomar en cuenta las siguientes características del mismo **(Coello, 1995)**:

- ✓ Su espacio de búsqueda debe estar delimitado dentro de un cierto rango.
- ✓ Debe poderse definir una función de aptitud que nos indique qué tan buena o mala es una cierta respuesta.
- ✓ Las soluciones deben codificarse de una forma que resulte relativamente fácil de implementar en la computadora.

**Dentro de las aplicaciones se encuentran (Goldberg, 1989)**:

- Optimización (estructural, de topologías, numérica, combinatoria, etc.)

- Aprendizaje de máquina (sistemas clasificadores)
- Bases de datos (optimización de consultas)
- Reconocimiento de patrones (por ejemplo, imágenes o letras)
- Generación de gramáticas (regulares, libres de contexto, etc.)
- Planeación de movimientos de robots
- Predicción

### 1.4.1. Algoritmos Evolutivos.

**Paradigmas de la computación evolutiva.**

El término "computación evolutiva" o "algoritmos evolutivos" engloba una serie de técnicas inspiradas en los principios de la teoría Neo-Darwiniana de la evolución natural. En términos generales, para simular el proceso evolutivo en una computadora se requiere:

- Codificar las estructuras que se replicarán (o sea, una estructura de datos que se utilice para almacenar a un "individuo").
- Operaciones que afecten a los "individuos" (típicamente se usa cruce y mutación).
- Una función de aptitud que nos indique qué tan buena es una solución con respecto a las demás.
- Un mecanismo de selección que implemente el principio de "supervivencia del más apto" de la teoría de Darwin.

Aunque hoy en día es cada vez más difícil distinguir las diferencias entre los distintos tipos de algoritmos evolutivos existentes, por razones sobre todo históricas, suele hablarse de tres paradigmas principales:

• Programación Evolutiva
• Estrategias Evolutivas
• Algoritmos Genéticos

Cada uno de estos paradigmas se originó de manera independiente y con motivaciones muy distintas, por lo que se procederá a describir brevemente a cada uno de ellos de forma totalmente independiente.

### 1.4.2. Programación Evolutiva

Lawrence J. Fogel propuso en 1960 una técnica denominada "Programación Evolutiva", en la cual la inteligencia se ve como un comportamiento adaptativo **(Fogel, 1966)**. La Programación Evolutiva enfatiza

los nexos de comportamiento entre padres e hijos, en vez de buscar emular operadores genéticos específicos (como en el caso de los Algoritmos Genéticos).

El algoritmo básico de la Programación Evolutiva es el siguiente:

- Generar aleatoriamente una población inicial.
- Se aplica mutación.
- Se calcula la aptitud de cada hijo y se usa un proceso de selección mediante torneo (normalmente estocástico) para determinar cuáles serán las soluciones que se retendrán.

La Programación Evolutiva es una abstracción de la evolución al nivel de las especies, por lo que no se requiere el uso de un operador de recombinación (diferentes especies no se pueden cruzar entre sí). Así mismo, usa selección probabilística.

Algunas aplicaciones de la Programación Evolutiva son **(Fogel, 1966)**:

• Predicción
• Generalización
• Juegos
• Control automático
• Problema del viajero
• Planeación de rutas
• Diseño y entrenamiento de redes neuronales
• Reconocimiento de patrones

### 1.4.3. Estrategias Evolutivas

Las Estrategias Evolutivas (EE) fueron desarrolladas en 1964 en Alemania para resolver problemas hidrodinámicos de alto grado de complejidad por un grupo de estudiantes de ingeniería encabezado por Ingo Rechenberg **(Bäck, 1966)**.

Rechenberg introdujo el concepto de población al proponer una estrategia evolutiva llamada $(\lambda + 1) - EE$ en la cual hay $\lambda$ padres y se genera un solo hijo, el cual puede reemplazar al peor padre de la población (selección extintiva) **(Rechenberg, 1973)**.

Los operadores de recombinación de las Estrategias Evolutivas pueden ser:

- Sexuales: el operador actúa sobre 2 individuos elegidos aleatoriamente de la población de padres.

- Panmíticos: se elige un solo padre al azar, y se mantiene fijo mientras se elige al azar un segundo padre (entre toda la población) para cada componente de sus vectores.

Algunas aplicaciones de las Estrategias Evolutivas son **(Schwefel, 1981)**:

- Problemas de rutas y redes
- Bioquímica
- Óptica
- Diseño en ingeniería
- Magnetismo

### 1.4.4. Estructura de los Algoritmos Genéticos.

El Algoritmo Genético Simple, también denominado Canónico, se representa en la Figura 2: **(Barán)**

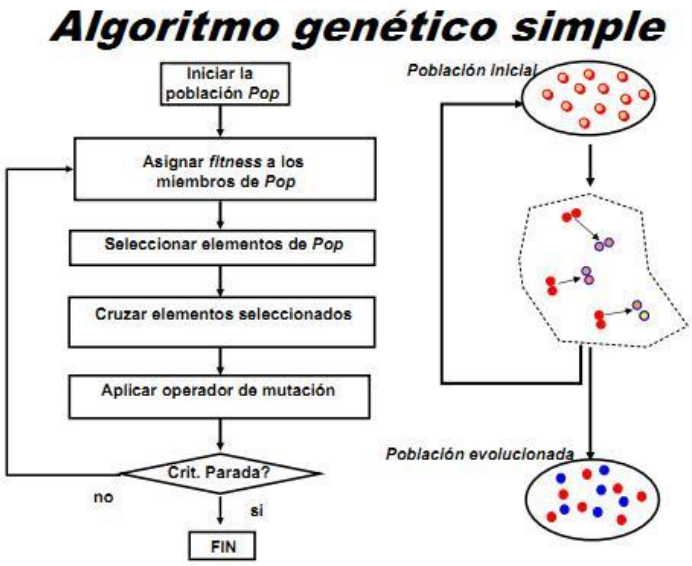


**Figura 2: Algoritmo Genético Simple**

Para realizar dicho proceso lo primero que se debe hacer es determinar cómo representar los parámetros del problema en un cromosoma y determinar la función objetivo para evaluar el desempeño de los individuos.

También se deben asignar las probabilidades de cruzamiento y mutación adecuadas para el problema y seguir los pasos que se describen a continuación **(Álvarez, y otros, 2010)**:

1. Se crea una población inicial aleatoria y se evalúa cada cromosoma de acuerdo con la función objetivo.
2. Se seleccionan los cromosomas con mejor valor *fitness* para ser reproducidos.
3. De los cromosomas seleccionados se escogen pares al azar para ser cruzados. Los cromosomas hijos forman la nueva generación.
4. Ocasionalmente se muta algún gen de acuerdo con la probabilidad de mutación.
5. Se repite el proceso de selección, cruzamiento y mutación de generación en generación hasta que se complete el número máximo de generaciones establecido, o se alcanza una solución suficientemente buena.

### 1.4.5. Codificación de las variables y tamaño de población.

En primer lugar, y quizás uno de los factores más determinantes a la hora de que el AG funcione correctamente, está en determinar la codificación más adecuada.

Como ya se ha dicho con anterioridad los individuos que componen la población de un AG son representaciones de soluciones posibles al problema, pero, en general, no son los valores de dichas soluciones. La explicación más sencilla a esto es que raramente se trate de un problema de optimización con una única variable o restricción. Los AG son aplicables tanto en problemas con una única restricción como con múltiples restricciones en cuyo caso muchas de las soluciones generadas serán posibles y, quizás, serían soluciones óptimas para algunas de las variables pero no serán válidas porque violarán algunas de las restricciones.

Representar esto mediante un único valor sería del todo imposible ya que provocaría que se perdiera gran cantidad de información. De manera que en general, cada uno de los individuos estará definido por un conjunto de valores representativos de cada una de las restricciones o variables del problema a los que denominaremos genes. Así pues, a imagen y semejanza de los seres vivos, cada uno de los individuos estará compuesto por un conjunto de genes que darán lugar a los cromosomas que definirán las características del individuo.

La siguiente figura representa un ejemplo de codificación entera:

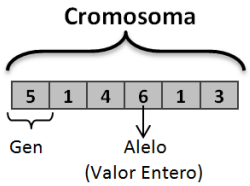


**Figura 3: Codificación entera**

La mayoría de las veces, una codificación correcta es la clave de una buena resolución del problema. Generalmente, la regla heurística que se utiliza es la llamada *regla de los bloques de construcción*, es decir, parámetros relacionados entre sí deben de estar cercanos en el cromosoma. En todo caso, se puede ser bastante creativo con la codificación del problema, teniendo siempre en cuenta la regla anterior. Esto puede llevar a usar cromosomas bidimensionales, o tridimensionales, o con relaciones entre genes que no sean puramente lineales de vecindad. En algunos casos, cuando no se conoce de antemano el número de variables del problema, caben dos opciones: codificar también el número de variables, fijando un número máximo, o bien, lo cual es mucho más natural, crear un cromosoma que pueda variar de longitud. Para ello, claro está, se necesitan operadores genéticos que alteren la longitud **(Merelo, 1997)**.
Normalmente, la codificación es estática, pero en casos de optimización numérica, el número de bits dedicados a codificar un parámetro puede variar, o incluso lo que representen los bits dedicados a codificar cada parámetro.

**Tamaño de la población.**

Una cuestión que se puede plantear es la relacionada con el tamaño idóneo de la población. Parece intuitivo que las poblaciones pequeñas corren el riesgo de no cubrir adecuadamente el espacio de búsqueda, mientras que el trabajar con poblaciones de gran tamaño puede acarrear problemas relacionados con el excesivo costo computacional. Goldberg efectuó un estudio teórico, obteniendo como conclusión que el tamaño óptimo de la población para ristras de longitud $l$, con codificación binaria, crece exponencialmente con el tamaño de la ristra **(Londoño, 2006)**.
Este resultado traería como consecuencia que la aplicabilidad de los AG en problemas reales sería muy limitada, ya que resultarían no competitivos con otros métodos de optimización combinatoria.

### 1.4.6. Población Inicial.

Habitualmente la población inicial se escoge generando ristras al azar, pudiendo contener cada gen uno de los posibles valores del alfabeto con probabilidad uniforme. Se podría preguntar qué es lo que

sucedería si los individuos de la población inicial se obtuviesen como resultado de alguna técnica heurística o de optimización local. La población inicial de un AG puede ser creada de muy diversas formas, desde generar aleatoriamente el valor de cada gen para cada individuo, utilizar una función ávida o generar alguna parte de cada individuo y luego aplicar una búsqueda local **(Londoño, 2006)**.

### 1.4.7. Función Objetivo.

Dos aspectos que resultan cruciales en el comportamiento de los AG son la determinación de una adecuada función de adaptación o función objetivo, así como la codificación utilizada. La función de adaptación debe ser diseñada para cada problema de forma específica. La regla general para construir una buena función objetivo es que ésta debe reflejar el valor del individuo de una manera real. Existen otros métodos para establecer la función objetivo como la técnica que se ha venido utilizando en el caso en que la computación de la función objetivo sea muy compleja, que es la denominada evaluación aproximada de la función objetivo. En algunos casos la obtención de $n$ funciones objetivo aproximadas puede resultar mejor que la evaluación exacta de una única función objetivo (supuesto el caso de que la evaluación aproximada resulta como mínimo $n$ veces más rápida que la evaluación exacta).

### 1.4.8. Operador de Selección.

El operador de selección es el encargado de transmitir y conservar aquellas características de las soluciones que se consideran valiosas a lo largo de las generaciones. El principal medio para que la información útil se transmita es que aquellos individuos mejor adaptados (mejor valor de función de evaluación) tengan más probabilidades de reproducirse. Sin embargo, es necesario también incluir un factor aleatorio que permita reproducirse a individuos que, aunque no estén muy bien adaptados, puedan contener alguna información útil para posteriores generaciones con el objetivo de mantener así también una cierta diversidad en cada población **(Londoño, 2006)**.

En la siguiente figura se muestran los operadores genéticos de selección:


**Figura 4: Operadores de selección**

- **Operadores de selección proporcional:** Los individuos son elegidos en función de su contribución de aptitud respecto al total de la población.
- **Operadores de selección mediante torneo:** Los individuos son elegidos en base a comparaciones directas entre ellos.
- **Operadores de selección de estado uniforme:** Se reemplazan los individuos menos aptos por nuevos individuos reproducidos a partir de la propia población.

La selección de un tipo u otro de operador dependerá de las características del problema a resolver y de la estrategia de búsqueda que se determinará para el AG. Algunos operadores buscan la eficiencia computacional, otros el número de veces que los peores o mejores individuos pueden ser seleccionados; no obstante, todos mantienen como objetivo principal determinar qué individuos participan en el proceso de reproducción o determinar la forma de construcción de la nueva población antes de pasar a la próxima generación **(Milian, y otros, 2011)**.

**Operadores de selección proporcional**

Los operadores que utilizan esta técnica de selección utilizan la función de evaluación para asignarle proporcionalmente a cada individuo un valor de aptitud acorde a la relación entre la calidad y el número de individuos presentes en la población. La principal desventaja que tienen es el costo computacional pues deben hacerse varias corridas entre los individuos de la población para evaluarlos y posteriormente seleccionar a los indicados.

- **Ruleta.**

La idea principal de esta técnica es la de asignarle a cada individuo una probabilidad de selección proporcional a su aptitud. La probabilidad asociada con su selección está dada por la siguiente función:

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i}$$

Una vez que se le ha asignado a cada individuo una probabilidad de selección utilizando esta función, se construye un modelo de una ruleta asignándole a cada individuo un espacio proporcional a su probabilidad. La selección se realiza haciendo girar la rueda de la ruleta tantas veces como individuos existan en la población y eligiendo los señalados por la ruleta. Esta selección permite que los mejores individuos sean escogidos con una mayor probabilidad, pero al mismo tiempo permite a los peores individuos ser elegidos, lo cual puede ayudar a mantener la diversidad de la población. Esto, en iteraciones tempranas del AG, puede ser visto como una ventaja pero puede convertirse posteriormente en una desventaja pues permite tener múltiples copias de las peores soluciones.

Otro problema de la selección de ruleta se presenta cuando existe una pequeña fracción de la población que posee una medida de desempeño excesivamente superior al resto. Esto provoca pérdida de diversidad y puede conducir a la convergencia prematura pues la mayor parte de los individuos seleccionados será una copia de los predominantes. En este caso es preferible utilizar otro operador de selección basado en *ranking* o en torneo **(Milian, y otros, 2011)**.

- **Sobrante Estocástico.**

Esta técnica permite mitigar uno de los problemas de la técnica de la ruleta relacionado con la selección de soluciones que no se esperan sean seleccionadas. Para ello utiliza una relación entre el valor de aptitud del individuo y la calidad promedio de la población para calcular la cantidad de copias que se espera del individuo en la nueva población. El valor de copias esperado está dado por la siguiente fórmula:

$$c_i = \frac{f_i N}{\sum_{i=1}^{N} f_i}$$

Una vez calculada la cantidad de copias esperadas por cada individuo se seleccionan determinísticamente los mismos según la parte entera del valor obtenido. El resto sobrante del redondeo se utiliza para rellenar probabilísticamente la población, con una de las dos variantes que son:

- Sin reemplazo: Cada sobrante se usa para sesgar el tiro de una moneda que determina si una cadena se selecciona de nuevo o no.
- Con reemplazo: Los sobrantes se usan para dimensionar los segmentos de una ruleta y se usa esta técnica de manera tradicional.

➢ **Muestreo Determinístico.**

Esta técnica es similar a la anterior; la única diferencia está dada en el tratamiento que realiza para seleccionar el resto de la población que debe de ser rellenada. Una vez calculada la cantidad de copias esperadas por cada individuo y seleccionados determinísticamente los mismos, construye una lista ordenada (de mayor a menor) a partir del resto sobrante del redondeo y selecciona los faltantes de la parte superior de la lista. Al ser similar al sobrante estocástico presenta los mismos problemas que éste **(Milian, y otros, 2011)**.

➢ **Universal Estocástico.**

Esta técnica reduce el costo computacional de las anteriores realizando en este caso la selección de forma progresiva hasta obtener todos los individuos necesarios. Para ello acumula los restos del redondeo en una suma iterada y utiliza este valor para priorizar a los individuos más aptos en la selección. No obstante, puede ocasionar convergencia prematura pues hace que los individuos más aptos se multipliquen rápidamente. Esta técnica no soluciona el problema de la selección proporcional que busca que el número de copias reales coincida con el número de copias esperadas.

➢ **Escalado.**

Esta técnica fue propuesta para mitigar los problemas de que individuos muy prometedores dominaran el proceso de selección. Su idea principal es la de realizar un proceso de mapeo del valor obtenido por la función de evaluación a un valor real que es utilizado en el proceso de selección.

Este procedimiento tiene dos objetivos fundamentales:

- Mantener una diferencia razonable entre los valores de aptitud de los individuos.
- Evitar que los individuos más prometedores predominen inicialmente el proceso de selección limitando su competencia pero permitiéndola más adelante.

El principal impedimento de esta técnica es que la mayoría de funciones de mapeo son dependientes del problema por lo que deben ser construidas y adaptadas para los mismos.
Baker introdujo la noción de *ranking* para poder solucionar este problema de las técnicas de escalado. El *Ranking* ignora el valor actual de la función de evaluación para realizar la selección y en su lugar establece un *ranking* entre los individuos para determinar su probabilidad de supervivencia **(Milian, y otros, 2011)**.

- ***Ranking*.**

Esta técnica consiste en calcular las probabilidades de selección atendiendo a la ordenación de la población por el valor de adaptación. Inicialmente se ordenan los individuos del más apto hasta el menos apto y seguidamente se asignan las probabilidades de selección teniendo en cuenta el *ranking* que tienen los individuos en el listado e ignorando los valores de aptitud de los mismos. Seguidamente se realiza la selección utilizando uno de los métodos tradicionales como la ruleta. Dos métodos son los que se utilizan comúnmente: el *ranking* lineal y el *ranking* exponencial.

**Operadores de Selección Mediante Torneo.**

La idea principal de este método consiste en realizar la selección en base a comparaciones directas entre individuos. Se baraja la población y después se hace competir a los cromosomas que la integran en grupos de tamaño predefinido (normalmente compiten en parejas) en un torneo del que resultarán ganadores aquéllos que tengan valores de aptitud más altos. Si se efectúa un torneo binario entonces la población se debe barajar dos veces. Esta técnica garantiza la obtención de múltiples copias del mejor individuo entre los progenitores de la siguiente generación (si se efectúa un torneo binario, el mejor individuo será seleccionado dos veces).
Existen dos versiones de selección mediante torneo:

- Determinística

En la versión determinística se selecciona al azar un número de individuos. Entre los individuos seleccionados se selecciona el más apto para pasarlo a la siguiente generación.

- Probabilística

La versión probabilística únicamente se diferencia en el paso de selección del ganador del torneo. En vez de escoger siempre el mejor se genera un número aleatorio. Si es menor que un parámetro (fijado para todo el proceso evolutivo) se escoge el individuo más apto y en caso contrario el menos apto.
Variando el número de individuos que participan en cada torneo se puede modificar la presión de selección. Cuando participan muchos individuos en cada torneo, la presión de selección es elevada y los peores individuos apenas tienen oportunidades de reproducción.
Un caso particular es el elitismo global. Se trata de un torneo en el que participan todos los individuos de la población con lo cual la selección se vuelve totalmente determinística. Cuando el tamaño del torneo es reducido, la presión de selección disminuye y los peores individuos tienen más oportunidades de ser seleccionados **(Milian, y otros, 2011).**

**Operadores de selección de estado uniforme.**
Estos tipos de operadores de selección son muy utilizados en Algoritmos Genéticos no generacionales en los que la solución del problema está representada por un conjunto de individuos y no individualmente. Normalmente este enfoque no se aplica a problemas de optimización aunque puede ser utilizado.

### 1.4.9. Operador de Cruce o *Crossover*.

Consiste en el intercambio de material genético entre dos cromosomas. El *crossover* es el principal operador genético, hasta el punto que se puede decir que no es un Algoritmo Genético si no tiene *crossover*, y, sin embargo, puede serlo perfectamente sin mutación, según descubrió Holland. El teorema de los esquemas confía en él para hallar la mejor solución a un problema combinando soluciones parciales.
Para aplicar el *crossover*, entrecruzamiento o recombinación, se escogen aleatoriamente dos miembros de la población. No pasa nada si se emparejan dos descendientes de los mismos padres; ello garantiza la perpetuación de un individuo con buena puntuación. Sin embargo, si esto sucede demasiado a menudo, puede crear problemas: toda la población puede aparecer dominada por los descendientes de algún gen que, además, puede tener caracteres no deseados. Esto se suele denominar en otros métodos de

optimización atranque en un mínimo local, y es uno de los principales problemas con los que se enfrentan los que aplican Algoritmos Genéticos **(Merelo, 1997)**.

El operador de cruce permite realizar una exploración de toda la información almacenada hasta el momento en la población y combinarla para crear mejores individuos. Dentro de los métodos habituales destacamos los siguientes:

**Cruce de un punto:** Es la más sencilla de las técnicas de cruce, conocida como *Single Point Crossover (SPX).* Se selecciona una posición en las cadenas de los progenitores y se intercambian los genes a la izquierda de esta posición.

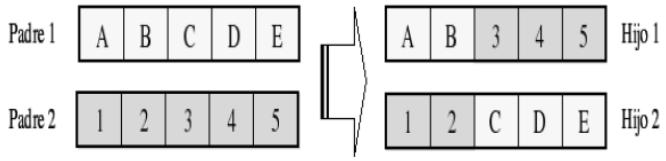


**Figura 5: Cruce de un punto**

**Cruce de n puntos:** Es una generalización del método anterior, conocido como *Double Point Crossover (DPX).* Se seleccionan varias posiciones (n) en las cadenas de los progenitores y se intercambian los genes a ambos lados de estas posiciones.

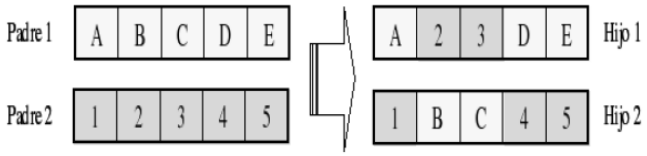


**Figura 6: Cruce en dos puntos**

**Cruce Uniforme:** Conocido como *Uniform Point Crossover (UPX).* El cruce uniforme es una técnica completamente diferente de las vistas anteriormente. Cada gen de la descendencia tiene las mismas probabilidades de pertenecer a uno u otro padre. Aunque se puede implementar de muy diversas formas, la técnica implica la generación de una máscara de cruce con valores binarios. Si en una de las posiciones de la máscara hay un 1, el gen situado en esa posición en uno de los descendientes se copia del primer padre. Si por el contrario hay un 0 el gen se copia del segundo padre. Para producir el segundo

descendiente se intercambian los papeles de los padres, o bien se intercambia la interpretación de los unos y los ceros de la máscara de cruce.

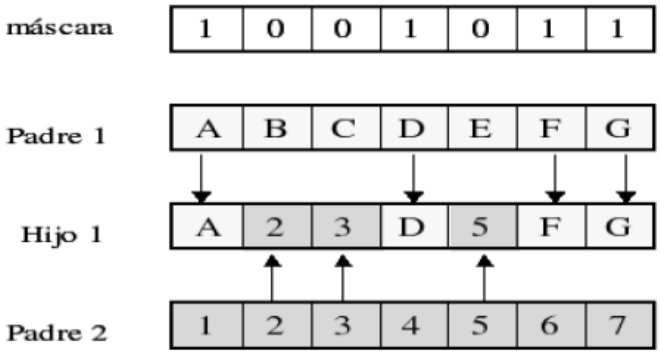


**Figura 7: Cruce uniforme**

**Cruces para permutación:** Existen varios métodos de cruzamiento para problemas donde, en la representación, importa el orden que tomen los valores de los genes, a continuación se explican algunos de ellos.

- **Cruce de mapeamiento parcial:** Conocido por *PMX (Partially Matched Crossover).*Toma una subsecuencia del genoma del padre y procura preservar el orden absoluto de los fenotipos, es decir, orden y posición en el genoma del resto del genoma lo más parecido posible de la madre. Se elige una subcadena central y se establece una correspondencia por posición entre las asignaciones contenidas en ellas. Cada hijo contiene la subcadena central de uno de los padres y el mayor número posible de asignaciones en las posiciones definidas por el otro padre.

Padre1 = (1 2 3 | 4 5 6 7 | 8 9)
Padre2 = (4 5 3 | 1 8 7 6 | 9 2)

Hijo'1 = (* * * | 1 8 7 6 | * *)
Hijo'2 = (* * * | 4 5 6 7 | * *)

Correspondencias: (1-4, 8-5, 7-6, 6-7)
Hijo1 = (***1-4*** 2 3 | 1 8 7 6 | ***8-5*** 9) = (4 2 3 | 1 8 7 6 | 5 9)

**Figura 8: Mapeamiento parcial**

- **Cruce de orden:** Conocido por *OX (Order Crossover).* Toma una subsecuencia del genoma del padre y procura preservar el orden relativo de los fenotipos del resto del genoma lo más parecido posible de la madre.

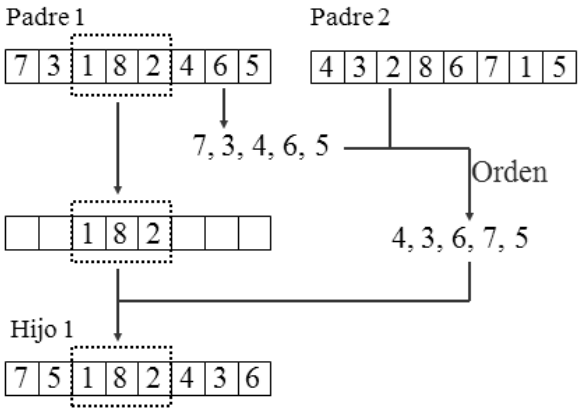


**Figura 9: Cruce de orden**

- **Cruce de ciclo**: Conocido por *CX (Cycle Crossover*). Se toma el primer gen del genoma del padre, poniéndolo en la primera posición del hijo y el primer gen del genoma de la madre, poniéndolo dentro del genoma del hijo en la posición que ocupe en el genoma del padre. El fenotipo que está en la posición que ocupa el gen del genoma del padre igual al primer gen del genoma de la madre se va a colocar en la posición que ocupe en el genoma del padre y así hasta rellenar el genoma del hijo.

Es una buena idea que, tanto la codificación como la técnica de cruce, se hagan de manera que las características buenas se hereden o, al menos, no sea mucho peor que el peor de los padres. En problemas en los que, por ejemplo, la adaptación es función de los pares de genes colaterales, el resultante del cruce uniforme tiene una adaptación completamente aleatoria **(Londoño, 2006)**.

### 1.4.10. Operador de Mutación.

En la evolución, una mutación es un suceso bastante poco común (sucede aproximadamente en una de cada mil replicaciones). En la mayoría de los casos las mutaciones son letales pero, en promedio, contribuyen a la diversidad genética de la especie. En un algoritmo genético tendrán el mismo papel y la misma frecuencia (es decir, muy baja) **(Merelo, 1997)**. Existen múltiples formas para llevar a vías de

hecho el proceso de mutación en un Algoritmo Genético y la selección de una u otra está asociada a las características del problema y sobre todo a la estrategia de codificación utilizada.

Todo operador de Mutación que se diseñe debe observar, como norma, las siguientes características:

- **Aptitud para alcanzar cualquier parte del espacio de búsqueda:** El operador debe ser capaz de alcanzar cualquier nodo del espacio de búsqueda sin importar su representación en el esquema de codificación utilizado. Así, no sería válido un operador de mutación que no sea capaz de alcanzar, por ejemplo, valores extremos del espacio de búsqueda. Esto garantiza que cualquier nodo del espacio de búsqueda tenga la misma probabilidad que el resto de ser, al menos, evaluado.
- **El tamaño de la mutación debe ser contralado:** La magnitud de los cambios introducidos por una iteración del proceso de mutación deben ser controlable. Del mismo modo la frecuencia con la que se efectúe el proceso debe tener una justa medida. Una elevada proporción de mutación provocaría una situación de búsqueda aleatoria, así como una proporción demasiado pequeña provocaría una convergencia prematura y por tanto poco confiable.
- **Cromosomas válidos como salida:** El operador debe estar diseñado de modo que el cromosoma resultante esté dentro del dominio definido por la codificación, garantizándose así que todo individuo generado por el algoritmo sea una posible solución.

Algunos de los operadores de mutación más usados son:

**Mutación binaria simple**: También denominada como *Flip Bit* o Cambio de Bit. Este es el más elemental de los operadores de mutación. Es aplicable a codificaciones del tipo binaria y simplemente cambia el valor de un gen por su opuesto.

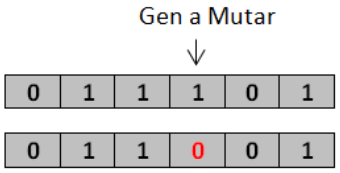


**Figura 10: Mutación binaria simple**

***Swap Mutation*:** Aplicable a codificaciones Basadas en el Orden. Consiste en seleccionar al azar dos posiciones del cromosoma e intercambiar sus valores. El proceso de selección de las posiciones se hace usando el método de la ruleta para garantizar igualdad de probabilidad para todas.

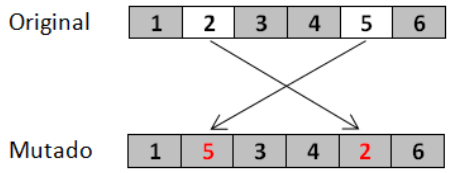


**Figura 11: *Swap Mutation***

**Mutación al Borde:** También denominada como *Boundary Mutation*. Basa su funcionamiento en cambiar el valor de una posición, elegida aleatoriamente, por su valor en un extremo del intervalo de definición de la variable. Los valores de los extremos pueden estar definidos por la propia naturaleza del problema o por medio de una matriz de extremos (*Bounds Matriz*).
En general para realizar este tipo de mutación:

1. Se elige una coordenada al azar del individuo seleccionado
2. Se elige la dirección de cambio, que puede ser hacia el mínimo el máximo
3. Se cambia el valor por el borde inferior o superior, según lo elegido en el paso anterior, del intervalo de definición

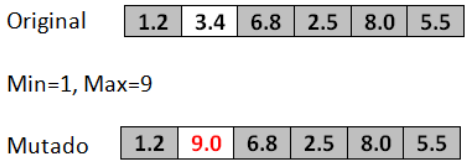


**Figura 12: Mutación al borde**

No hace falta decir que no conviene abusar de la mutación. Es cierto que es un mecanismo generador de diversidad y, por tanto, la solución cuando un algoritmo genético está estancado, pero también es cierto que reduce el algoritmo genético a una búsqueda aleatoria. Siempre es más conveniente usar otros mecanismos de generación de diversidad, como aumentar el tamaño de la población o garantizar la aleatoriedad de la población inicial **(Merelo, 1997)**.

### 1.4.11. Remplazo de la población y condición de parada.

Cada vez que se aplica el operador de cruce nos encontramos con un número de nuevos individuos (la descendencia) que se han de integrar en la población para formar la siguiente generación. Esta operación se puede hacer de diversas formas, pero en general existen tres métodos fundamentales para realizar el reemplazo:

- Cuando el número de individuos llega a una cantidad, se elimina un subconjunto de la población conteniendo a los individuos peor adaptados.
- Cada vez que se crea un nuevo individuo, en la población se elimina el peor adaptado para dejar su lugar a este nuevo individuo.
- Cada vez que se crea un nuevo individuo, en la población se elimina aleatoriamente una solución, independientemente de su adaptación.

En cuanto al criterio de parada, generalmente viene determinado por criterios sencillos, como un número máximo de generaciones o un tiempo máximo de resolución, o más eficientemente por estrategias relacionadas con indicadores del estado de evolución de la población, como por la pérdida de diversidad dentro de la población o por no haber mejora en un cierto número de iteraciones, siendo por lo general una condición mixta lo más utilizado, es decir, limitar el tiempo de ejecución a un número de iteraciones y tener en cuenta algún indicador del estado de la población para considerar la convergencia antes de alcanzar tal limitación **(Londoño, 2006)**.

## 1.5. Introducción a la Programación Distribuida.

En programación distribuida, existe un conjunto de ordenadores conectados por una red que son usados colectivamente para realizar tareas distribuidas. Por otro lado en los sistemas paralelos, la solución a un problema importante es dividida en pequeñas tareas que son repartidas y ejecutadas para conseguir un alto rendimiento **(López, 2006)**.

Las aplicaciones distribuidas funcionan siguiendo un modelo cliente/servidor. Uno o más servidores crean unos objetos locales y luego atienden peticiones de acceso sobre esos objetos provenientes de clientes situados en lugares remotos de la red.

De forma general, se puede decir que las necesidades de una aplicación distribuida son:

- Mecanismos para localizar los objetos en la red.
- Comunicación con los objetos remotos.

- Mecanismos de intercambio de información (paso de parámetros).

Un sistema distribuido es aquel cuyos componentes están en distintos ordenadores conectados en red y se comunican y coordinan a través del paso de mensajes. Implica:

- Concurrencia de los componentes.
- Carencia de un reloj global (no se pueden sincronizar perfectamente todos los nodos)
- Fallos independientes de los componentes.

¿Para qué montar un sistema distribuido?

- Compartir un recurso.
- Aprovechar la potencia de varios nodos.
- Aprovechar los recursos móviles.

**Características de la Programación Distribuida.**

Es un paradigma de programación enfocado en desarrollar sistemas distribuidos, abiertos, escalables, transparentes y tolerantes a fallos. Este paradigma es el resultado natural del uso de las computadoras y las redes. La programación distribuida típicamente cae en alguna de las varias arquitecturas básicas o arquitecturas: cliente-servidor, objetos distribuidos, entre otras.

**Ventajas**

Con respecto a Sistemas Centralizados:

- Una de las ventajas de los sistemas distribuidos es la economía, pues es mucho más barato añadir servidores y clientes cuando se requiere aumentar la potencia de procesamiento.
- El trabajo en conjunto. Por ejemplo: en una fábrica de ensamblado, los robots tienen sus CPUs diferentes y realizan acciones en conjunto, dirigidos por un sistema distribuido.
- Tienen una mayor confiabilidad. Al estar distribuida la carga de trabajo en muchas máquinas la falla de una de ellas no afecta a las demás y el sistema sobrevive como un todo.
- Capacidad de crecimiento incremental. Se puede añadir procesadores al sistema incrementando su potencia en forma gradual según sus necesidades.

Con respecto a PCs Independientes:

- Se pueden compartir recursos, como programas y periféricos, muy costosos. Ejemplo: Impresora Láser, dispositivos de almacenamiento masivo, entre otros.

- Al compartir recursos, satisfacen las necesidades de muchos usuarios a la vez. Ejemplo: Sistemas de reservas de aerolíneas.
- Se logra una mejor comunicación entre las personas. Ejemplo: el correo electrónico.
- Tienen mayor flexibilidad, la carga de trabajo se puede distribuir entre diferentes ordenadores.

### 1.6. Algoritmos Genéticos Distribuidos

- **Modelo de islas**

Conocidos como **Sub-poblaciones estáticas con migración** o **Algoritmos Genéticos de grano grueso**. En éstos, cada proceso supone una instancia de nuestro algoritmo, que ejecuta de forma secuencial el mismo problema. Cada población local es inicializada independientemente, de manera que cada isla partirá de poblaciones distintas y obtendrá resultados diferentes. Al final de la ejecución, las poblaciones de cada una de las islas son combinadas para conformar una población global de la que seleccionaremos el mejor individuo. Este enfoque tiene la ventaja de que es mucho más resistente a óptimos locales durante el proceso de búsqueda, debido a que cada población es teóricamente independiente de las demás. Una característica muy interesante de este modelo es la introducción del concepto de migración. La migración entre islas es un artefacto que proporciona el intercambio periódico de individuos, posiblemente los mejores, entre las islas de acuerdo a una topología de interconexión definida **(Pit, 1995; Alba, y otros, 2001)**.

**Topologías de interconexión (Miki, y otros, 1999):**

- **Anillo:** cada subpoblación envía sus $p_i$ mejores individuos a una población vecina, efectuándose la migración en un único sentido de flujo.
- **Estrella:** existe una subpoblación que es seleccionada como maestra (aquella que tiene mejor media en el valor de la función objetivo), siendo las demás consideradas como esclavas. Todas las subpoblaciones esclavas mandan sus $p_i$ mejores individuos ($p_i \geq 1$) a la subpoblación maestra, la cual a su vez manda sus $p_j$ mejores individuos $(p_j \geq 1)$ a cada una de las subpoblaciones esclavas.

En el concepto de migración deben tenerse en cuenta los elementos siguientes **(Manderick, y otros, 1991)**:

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

➢ **Variante asíncrona maestro-esclavo**

El proceso maestro reparte las tareas (subgrupos de la población original) a los procesos esclavos disponibles los cuales ejecutan el algoritmo secuencial y devuelven la mejor solución encontrada. El proceso maestro es el encargado de notificar los cambios en la mejor solución global encontrada a los procesos esclavos, estos a su vez le harán saber al proceso maestro si están atrapados en un óptimo local. En esta variante no existe comunicación entre las diferentes subpoblaciones.

### 1.7. Análisis de soluciones existentes.

A escala mundial se han desarrollado soluciones a la problemática que da origen a esta investigación. En el presente epígrafe se hace un análisis de dichas soluciones para intentar buscar apoyo en alguna de ellas que sirva de base para proponer la solución de esta investigación o comprobar si existe alguna semejante que permita enfrentar al problema planteado. Ejemplo de estas soluciones son:

**JEO: Java Evolving Objects.**

El desarrollo y evolución de clasificadores que permitan la clasificación de señales de encefalograma (EEG) es una parte integral del proyecto de construcción de un interfaz cerebro-computadora. Actualmente, este trabajo se realiza mediante la biblioteca JEO desarrollada por Arenas **("JEO: Java Evolving Objects", 2002)** dentro del marco del proyecto europeo DREAM **("DREAM Distributed Resources Evolutionary Algorithm Machine", 2000)**. JEO está diseñada para trabajar sobre DRM, una red P2P para la ejecución distribuida de aplicaciones java.

**ECJ.**

Es una plataforma de desarrollo de algoritmos evolutivos desarrollada en el Laboratorio de Computación Evolutiva ECLab de la George Mason University. De su página web (http://cs.gmu.edu/~eclab/projects/ecj/): “Ha sido diseñada para ser altamente flexible, con la mayoría de clases y sus ajustes determinados dinámicamente durante la ejecución mediante un archivo de parámetros proporcionado por el usuario. Todas las estructuras del sistema están preparadas para ser fácilmente modificables”.

Otros de los estudios encontrados sobre el tema en la Universidad de Ciencias Informáticas es el **trabajo de diploma de Rigoberto Leander Salgado Reyes** en el cual el autor desarrolla una biblioteca paralela de algoritmos evolutivos aplicados a problemas computacionalmente costosos. En dicho trabajo el autor hace un estudio de los algoritmos de optimización e implementa una biblioteca en lenguaje C de forma paralela para el trabajo con algoritmos evolutivos.
Dichos estudios no resuelven la problemática de la investigación debido a que se necesita una biblioteca de propósito general y eficiente, implementada en lenguaje Java y de fácil integración con la plataforma T-Arenal.

## 1.8. Herramientas y Tecnologías a utilizar.

### 1.8.1. Entorno de Desarrollo Integrado (IDE)

**Eclipse**

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).
El IDE Eclipse es, únicamente, una de las herramientas que se engloban bajo el denominado Proyecto Eclipse. La arquitectura de *plugins* de Eclipse permite, además de integrar diversos lenguajes sobre un mismo IDE, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces, ayuda en línea para bibliotecas, etc.

### 1.8.2. Lenguaje de Programación Java.

Java es un lenguaje simple, posee una curva de aprendizaje muy rápida. Resulta relativamente sencillo escribir *applets* interesantes desde el principio. Todos aquellos familiarizados con C++ encontrarán que Java es más sencillo, ya que se han eliminado ciertas características, como los punteros. Debido a su semejanza con C y C++ y dado que la mayoría de la gente los conoce aunque sea de forma elemental, resulta muy fácil aprender Java. **(Marañón, 1997-1999)**

**Características de Java**

Orientado a objetos: Java fue diseñado como un lenguaje orientado a objetos desde el principio. Los objetos agrupan en estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manipulan esos datos. La tendencia del futuro, a la que Java se suma, apunta hacia la programación orientada a objetos, especialmente en entornos cada vez más complejos y basados en red.

Distribuido: Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir *sockets* y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

Robusto: Java fue diseñado para crear *software* altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores (la aritmética de punteros), ya que se ha prescindido por completo de los punteros y la recolección de basura elimina la necesidad de liberación explícita de memoria.

Portable: La indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas.

### 1.8.3. Herramienta *Case*.

Las Herramientas *CASE* son un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de *software* y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un *Software*. Actualmente existe una gran variedad de herramientas CASE para el proceso de desarrollo de *software*, por lo que seleccionar alguna se convierte en una difícil tarea.

***Visual Paradigm***

El *Visual Paradigm* es una herramienta colaborativa, es decir, soporta múltiples usuarios trabajando sobre el mismo proyecto, constituye una herramienta profesional para el modelado UML que soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño, construcción, pruebas y despliegue. Posibilita el modelado de base de datos, requerimientos, proceso de negocio, permite realizar todo tipo de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Características que posee *Visual Paradigm*:

- Producto de calidad.
- Soporta aplicaciones web.
- Generación de código para Java y exportación como HTML.
- Fácil de instalar y actualizar.
- Licencia: Gratuita y Comercial.
- Compatibilidad entre ediciones.

Posibilita la representación gráfica de los diagramas permitiendo ver el sistema desde diferentes perspectivas, como el de componentes, despliegue, secuencia, casos de uso, clase, actividad, entre otros. Además, se centra en cómo los componentes del sistema interactúan entre ellos, sin entrar en detalles excesivos; también permite ver las relaciones entre los componentes del diseño y mejora la comunicación entre los miembros del equipo usando un lenguaje gráfico. Brinda la posibilidad de generar código a partir de los diagramas, para plataformas como .Net, Java y PHP, así como obtener diagramas a partir del código **(Visual Paradigm International Ltd.).**

### 1.8.4. Plataforma de tareas distribuidas T-Arenal

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

usuario final como un solo ordenador y todos los servidores serán gestionados a través de un servidor central.

### 1.9. Conclusiones

En la elaboración de este capítulo se abordaron conceptos generales que sirvieron de guía para lograr un correcto dominio de la investigación, tratando de proporcionar de forma breve y concisa un panorama general de lo que son los Algoritmos Genéticos y su utilidad. Con la búsqueda y estudio de algunas soluciones existentes de bibliotecas que permitan la solución de problemas de optimización computacionalmente costosos mediante Algoritmos Genéticos, sirvió de gran apoyo para la búsqueda de la solución a nuestro problema científico. Se definieron además las tecnologías y herramientas fundamentales que se utilizarán para lograr que la biblioteca cuente con las tecnologías adecuadas para un mejor desarrollo de la misma.

# Capítulo 2: Diseño e Implementación

## 2.1 Introducción

En el presente capítulo se describen los elementos fundamentales relativos al diseño e implementación de la biblioteca. Se presenta el diseño propuesto y la implementación del mismo. Se describen las principales clases y sus funcionalidades, así como las pautas que definen su funcionamiento básico.

## 2.2 Diseño de la biblioteca

La biblioteca desarrollada es de propósito general, permitiendo resolver diferentes tipos de problemas de optimización. Para ello, cada problema debe ser adaptado de forma tal que ésta pueda darle solución, con este fin, el usuario debe dar ciertas informaciones sobre el problema en cuestión. La biblioteca brinda la opción de una ejecución secuencial así como una distribuida utilizando la plataforma de cálculo distribuido T-Arenal.

La utilización del paradigma orientado a objetos para la concepción de la biblioteca permitió las siguientes ventajas:

• **Reusabilidad:** una misma implementación de un algoritmo permite resolver una amplia gama de problemas muy diferentes entre sí.

• **Facilidad de ampliación y de modificación:** se dispone de varios mecanismos de ampliación como son la herencia, la redefinición de métodos, polimorfismo, que permiten ampliar la biblioteca con relativa facilidad. El encapsulamiento en clases de los elementos que conforman la biblioteca permite modificar uno de ellos sin afectar al resto.

• **Facilidad de aprendizaje y transparencia de uso:** la biblioteca oculta la implementación del algoritmo, con lo que el usuario final no necesita conocerla, únicamente rellenar las clases concretas al problema para utilizarla.

Para modelar un problema de optimización mediante los AG y para que sea ejecutado por la biblioteca, debemos definir primeramente como estará conformado el gen, eslabón fundamental en la modelación, la biblioteca provee una interfaz, *IGene* que nos especifica cómo deben ser definidos los mismos. Como

muestra la Figura 13, los genes componen a los cromosomas (*IChromosome*) y estos a su vez componen a la población (*Population*) que será evolucionada a través del genotipo (*Genotype*).

Luego debemos definir los operadores (*IGeneticOperator*) y selectores (*INaturalSelector*), los cuales serán encargados de evolucionar la población de cromosomas y se encontrarán en la configuración del algoritmo (*Configuration*). Además, hay que definir la función objetivo del problema (*GAFitnessFunction)* que también estará incluida en la configuración. En el epígrafe 2.3 se explican con más detalle las clases fundamentales y sus principales funcionalidades.

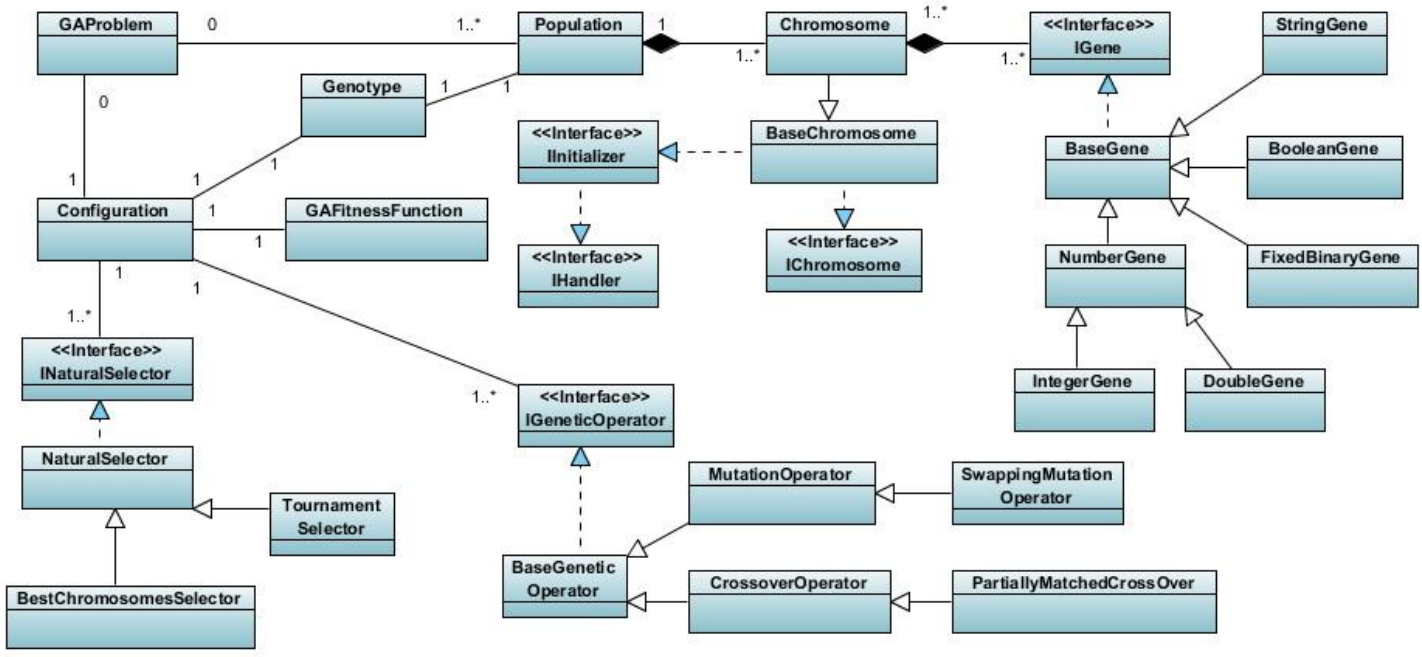


**Figura 13: Diagrama UML de la biblioteca**

## 2.3 Implementación de la biblioteca.

En este epígrafe se presentan las principales clases de la biblioteca así como las principales funcionalidades de las mismas.

### 2.3.1 Clase *GAProblem*.

La clase *GAProblem* representa el problema de optimización que se quiere resolver, por lo que debe contener los elementos básicos que definen el problema específico. Cuenta con un atributo *Configuration*

que tendrá la configuración general del algoritmo genético, un atributo booleano *ramdomInitialPopulation* que definirá si la población inicial será generada aleatoriamente o no y el atributo *Population* que representa la población que pasará por el proceso evolutivo.

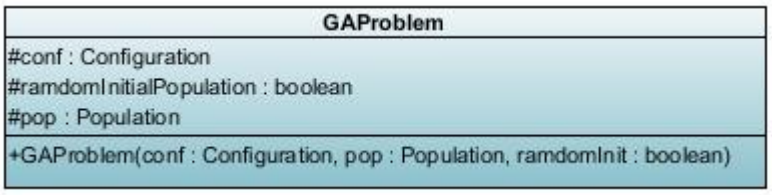


**Figura 14: UML *GAProblem***

### 2.3.2 Función de aptitud

La función de aptitud determina cuán óptima es una determinada solución. Es una parte indispensable del algoritmo que debe ser definido por el desarrollador. Dicha función debe ser implementada heredando de la clase abstracta *GAFitnessFunction*. Para determinar la aptitud de una solución la clase cuenta con el método *evaluate(T a_subjet)* que recibe un individuo por parámetros y devuelve el valor de aptitud del mismo.

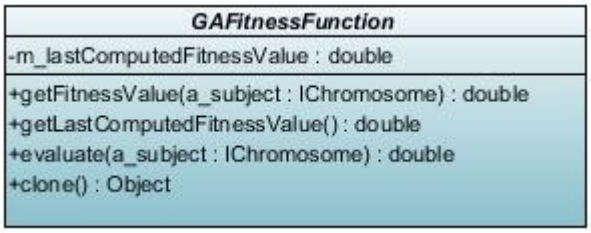


**Figura 15: UML *GAFitnessFunction***

### 2.3.3 Configuración

Para definir la configuración de nuestro AG se implementa la clase *Configuration*. Dicha clase cuenta con todos los parámetros necesarios para llevar a cabo el proceso evolutivo del AG, como por ejemplo la función objetivo, los selectores naturales y los operadores genéticos. Dicha clase además determina cómo van a estar constituidos los cromosomas y la población y contiene un atributo *IFactory* que es el encargado de crear e inicializar los cromosomas, así como un atributo *IBreeder* que es el encargado de "criar" la población y evolucionarla a través del genotipo. La biblioteca cuenta con la clase

*DefaultConfiguration* que hereda de *Configuration* y contiene una configuración por defecto recomendada, que puede ser modificada según sea necesario.


**Figura 16: UML *Configuration***

### 2.3.4 Genes.

En un algoritmo genético los genes representan un componente fundamental de la posible solución, el cromosoma. Los genes pueden contener cualquier tipo de información o dato (alelo) por lo que para cumplir dicha característica se diseñó la interfaz *IGene* que contiene las funcionalidades generales de los genes. Dicha interfaz es implementada por la clase abstracta *BaseGene*, la cual es ajena al tipo de dato que contendrá. La biblioteca contiene las siguientes implementaciones de genes que pueden ser usados según el problema que se quiera resolver.

- *NumberGene*
    - *IntegerGene*

- *DoubleGene*
- *BooleanGene*
- *StringGene*
- *BinaryGene*

A continuación se muestra el UML de una de las implementaciones contenidas en la biblioteca:

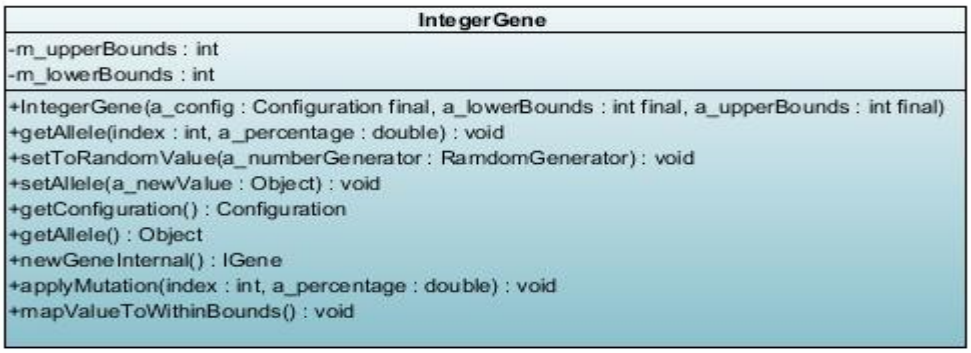


**Figura 17: UML *IntegerGene***

### 2.3.5 Cromosomas.

Un elemento fundamental en la resolución de problemas de optimización, lo constituye la representación de las soluciones. Los cromosomas constituyen dicho elemento dentro de un AG ya que éstos contienen la información genética (genes) y representan las posibles soluciones al problema. La biblioteca cuenta con la interfaz *IChromosome* que contiene las principales funcionalidades llevadas a cabo por esta estructura, esta interfaz es implementada por la clase abstracta *BaseChromosome* de la cual hereda *Chromosome*.

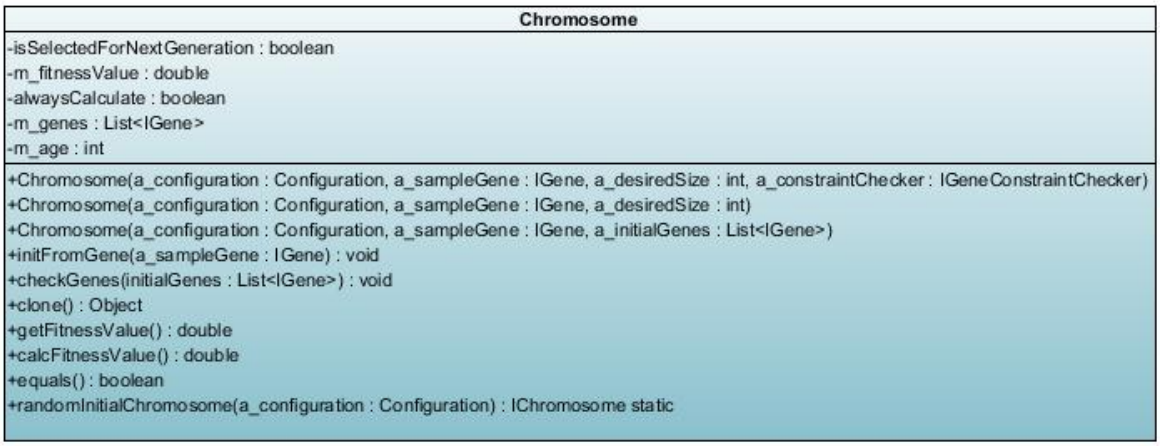


**Figura 18: UML *Chromosome***

### 2.3.6 Población.

En un AG la población es aquella que contiene a los individuos (cromosomas) y sobre la cual se realiza todo el proceso evolutivo. Para dicho propósito se implementa en la biblioteca la clase *Population*. Para facilitar el trabajo del desarrollador a la hora de obtener las mejores soluciones se han implementado en dicha clase los métodos *sortByFitness()*, el cual ordena la lista de cromosomas según su aptitud y el método *determineFittestChromosomes(int a_numberOfChromosomes)*, el cual devuelve una lista con la cantidad deseada de los cromosomas con mejor valor de aptitud.

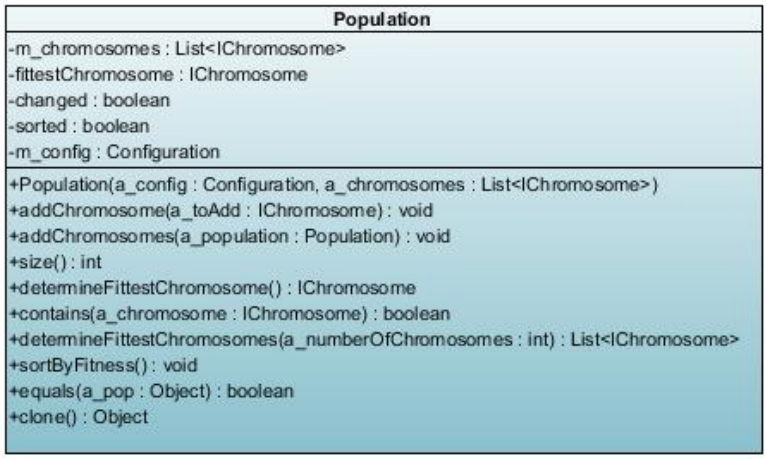


**Figura 19: UML *Population***

### 2.3.7 Clase *Genotype.*

La clase *Genotype* representa una población de tamaño fijo y es la encargada de evolucionar dicha población. Esta clase puede ser instanciada vía constructor o mediante el método estático *randomInitialGenotype()* que además es el encargado de crear una población inicial aleatoria. El método principal del genotipo es *evolve()*, método encargado de evolucionar la población.

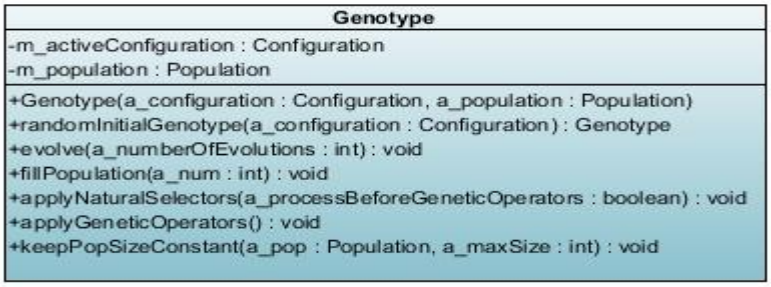


**Figura 20: UML *Genotype***

### 2.3.8 Selección natural.

La selección natural es la encargada de determinar cuáles individuos se reproducen y, después del proceso evolutivo de una generación, cuáles pasan a la siguiente generación en caso de tener que mantener la cantidad de individuos de la población constante. La biblioteca cuenta con la interfaz *INaturalSelector* que es implementada por la clase abstracta *NaturalSelector* de la cual heredan algunas implementaciones específicas. A continuación se presenta el UML de la implementación de selección por torneo.

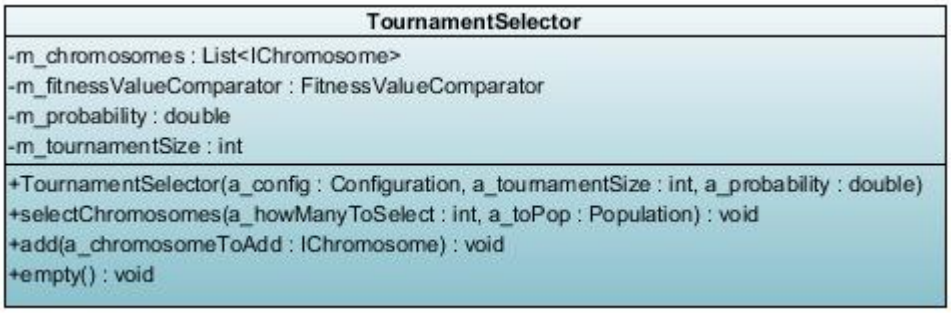


**Figura 21: UML *TournamentSelector***

### 2.3.9 Operadores genéticos.

Un operador genético representa una operación que tiene lugar sobre una población de individuos durante el proceso de evolución. En la biblioteca los operadores genéticos representan al cruzamiento y la

mutación. La interfaz que representa dichos operadores es *IGeneticOperator* que es implementada por la clase abstracta *BaseGeneticOperator* de la cual heredan las implementaciones de cruzamiento y mutación.

A continuación se presentan los UML de una implementación de cruzamiento y otra de mutación.

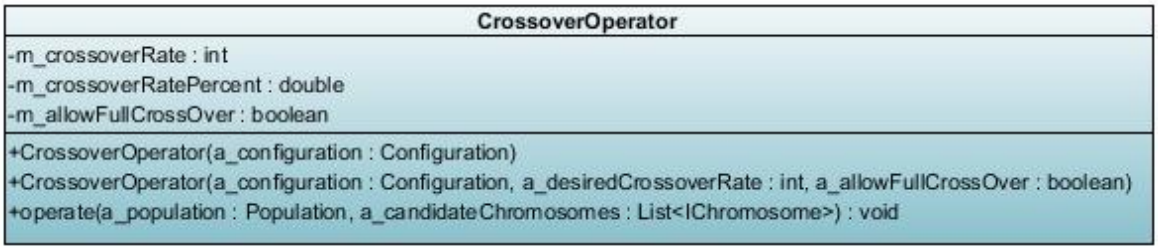


**Figura 22: UML *CrossoverOperator***

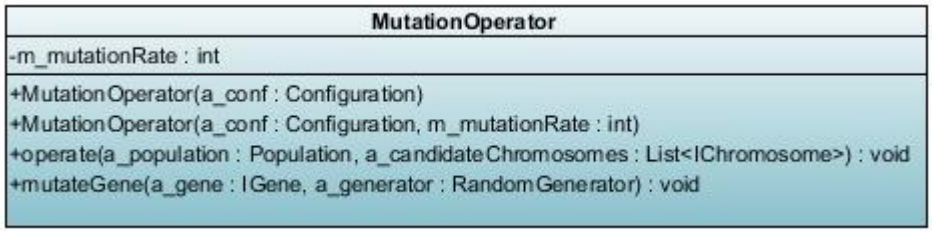


**Figura 23: UML MutationOperator**

### 2.4 Integración con la plataforma T-Arenal.

La ejecución distribuida del algoritmo se realiza sobre la plataforma T-Arenal; para ello el usuario debe heredar de las clases *DataManager* y *Task*. Dichas clases son abstractas y contienen los métodos necesarios para la ejecución distribuida de cualquier problema. El desarrollador debe redefinir dichos métodos a fin de establecer su aplicación distribuida.

- **Clase *DataManager*.**

El propósito de la clase *DataManager* es generar todos las unidades que se enviarán a los clientes, procesar todos los resultados enviados por los clientes, ajustar la granularidad (tamaño) de las unidades de trabajo, generar la información sobre el estado en que se encuentra la tarea en un instante determinado y analizar el cómputo distribuido **(Jacas)**. Con respecto al AG dicha clase es la encargada de dividir la población en subpoblaciones más pequeñas, enviarlas a las unidades de procesamiento donde será ejecutado el proceso evolutivo de forma secuencial y, después de procesadas las subpoblaciones, recibir los resultados y determinar las mejores soluciones. La clase *GADataManager* además cuenta con

un atributo *IIslandModel*, interfaz encargada de realizar la migración entre las subpoblaciones. La biblioteca contiene dos implementaciones de dicha interfaz, la clase *RingIslandModel* que realiza la migración siguiendo una topología de anillo y la clase *StarIslandModel*, la cual migra a las subpoblaciones según la topología de estrella. Ver epígrafe 1.6.

- **Clase *Task***

La instancia de la clase *Task* correspondiente a una tarea en particular, se ejecuta en cada uno de los clientes pertenecientes al servidor de peticiones que atiende o colabora en la realización de dicha tarea. Su propósito es procesar las unidades de trabajo generadas por el método *generateWorkUnit* del *DataManager*. En esta clase sólo el método *processUnit* debe ser redefinido con el fin de especificar el algoritmo a ejecutar **(Jacas)**. La clase *GATask* es la encargada de ejecutar el algoritmo genético en cada uno de los clientes de forma secuencial. Dicha clase recibe la subpoblación que debe pasar por el proceso evolutivo y devuelve al *DataManager* la población ya evolucionada.

## 2.5 Conclusiones

En el desarrollo de este capítulo se diseñaron e implementaron las clases necesarias para la ejecución de un AG, a partir de dicho diseño e implementación se logran las principales funcionalidades necesarias que hacen que la biblioteca sea de propósito general, eficiente, en lenguaje Java, y capaz de resolver problemas de optimización computacionalmente costosos, sobre la plataforma de cálculos distribuidos T-Arenal.

Con la estructura de clases interfaz, clase abstracta e implementaciones el desarrollador puede utilizar las clases ya implementadas y heredar de las clases abstractas en el caso que se necesite redefinir algo en específico que no resuelven las clases contenidas en la biblioteca, logrando gran flexibilidad y facilidad de implementación.

# Capítulo 3: Resultados y discusión

## 3.1 Introducción

En el presente capítulo se presentan y discuten los resultados obtenidos en las pruebas realizadas a la biblioteca mediante la modelación y resolución de problemas de asignación cuadrática. Se brinda una panorámica de los problemas de asignación cuadrática para lograr un entendimiento de la modelación del problema. Se presentan las soluciones secuenciales y distribuidas, así como los análisis comparativos en cuanto a los resultados obtenidos referidos a los parámetros de tiempo de ejecución y calidad de la solución. Finalmente se muestran algunas consideraciones acerca del cumplimiento de la hipótesis de investigación planteada.

## 3.2 Problema de Asignación Cuadrática (QAP)

El Problema de Asignación Cuadrática, conocido como QAP por sus siglas en inglés, es un problema importante en la teoría y en la práctica. El QAP puede ser mejor descrito como el problema de asignar un conjunto de instalaciones en un conjunto de localizaciones, teniendo las distancias entre las localizaciones y los flujos entre las instalaciones. La meta es posicionar las instalaciones en las localizaciones de tal forma que la suma de los productos entre flujos y distancias esté minimizada.

Más formalmente, teniendo $n$ instalaciones y $n$ localizaciones, dos matrices $n * n$, $A = [a_{ij}]$ y $B = [b_{rs}]$, donde $a_{ij}$ es la distancia entre las localizaciones $i$ y $j$ y $b_{rs}$ es el flujo entre las instalaciones $r$ y $s$ , siendo la función objetivo:

$$f(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} * a_{\pi_i \pi_j}$$

Donde $\pi_i$ brinda la ubicación de la instalación $i$ en la solución $\pi \in S(n)$, entonces la meta del QAP es encontrar una asignación de instalaciones en localizaciones tal que minimice la función objetivo.

El QAP es un problema NP-Duro y es considerado uno de los problemas de optimización combinatoria más complejos en la práctica **(P-complete approximation problems, 1976)**.

## 3.3 Modelación de QAP

Para modelar el problema mediante la biblioteca se utilizaron clases contenidas en la misma y se definieron otras específicas para QAP, dichas clases se encuentran en el paquete *QAPImp* y pueden ser usadas para resolver instancias de QAP, descargadas del sitio oficial de QAPLib **(Burkard, y otros, 2002)**. La codificación utilizada es la entera donde cada gen tiene un valor entre 1 y el tamaño de la instancia a resolver y el cromosoma contiene tantos genes como el tamaño de la instancia, de esta forma un gen con valor 3 ubicado en la posición 5 del cromosoma representa que la instalación 3 se ubica en la localización 5. Las clases utilizadas son las siguientes:

- ✓ Configuración: *QAPDefaultConfiguration*
- ✓ Función Objetivo: *QAPFitnessFuntion*
- ✓ Evaluador de aptitud: *MinFitnessEvaluator*
- ✓ Cromosomas: *PermutationsChromosome*
- ✓ Genes: *IntegerGene*
- ✓ Selección: *BestChromosomeSelector*
- ✓ Cruzamiento: *PartiallyMatchedCrossover*
- ✓ Mutación: *SwappingMutationOperator*

Tanto para las pruebas secuenciales como para las distribuidas se utilizaron los siguientes parámetros:

- ✓ Probabilidad de Selección: 0.9
- ✓ Probabilidad de Cruzamiento: 0.7
- ✓ Probabilidad de Mutación: 0.1

Además, en las pruebas se variaron los parámetros de tamaño de la población y cantidad de generaciones para poder observar el comportamiento de la biblioteca con dichas variaciones.

Para las pruebas se utilizaron las siguientes instancias de QAP descargadas del sitio oficial QAPLib:

- **Instancia 1:** sko100a.dat (100 x 100).
- **Instancia 2:** tai256c.dat (256 x 256)

### 3.4 Evaluación de la biblioteca de forma secuencial.

Las pruebas fueron realizadas en una PC dedicada con procesador Intel (R) Pentium (R) 4 a 3.00 GHz y memoria principal de 1 GB de RAM y sistema operativo Linux.

En las pruebas se varió el tamaño de la población y el número de generaciones, para ambas instancias. Los resultados demuestran que, al aumentar el tamaño de la población, aumenta considerablemente el tiempo y las soluciones obtenidas poseen valores de adaptación similares, mientras que si se aumenta el número de generaciones, también aumenta el tiempo de ejecución pero se obtiene una solución con un valor de adaptación considerablemente mejor.

La Figura 24 y la Figura 25 muestran los valores de aptitud obtenidos en ambas instancias con poblaciones de 500 y 1000 individuos y con 100 y 250 generaciones. Se puede observar como las mejores soluciones se obtuvieron con población de 1000 y 250 generaciones.

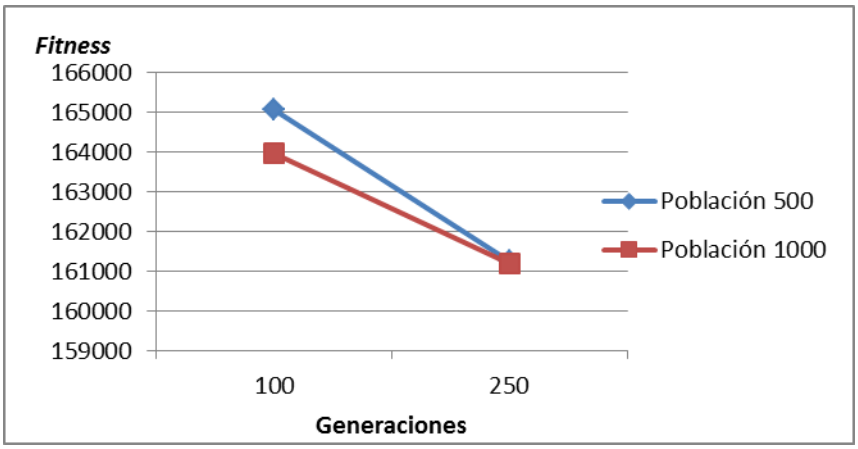


**Figura 24: *Fitness* prueba secuencial con instancia 1.**

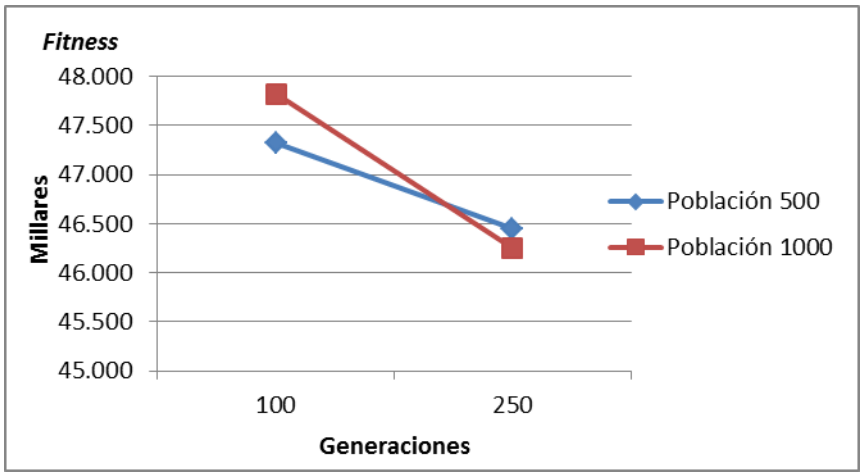


**Figura 25: *Fitness* prueba secuecial con instancia 2.**

En cuanto al tiempo obtenido, como muestra la Figura 26 y la Figura 27, aumenta proporcionalmente al tamaño de la población, al número de generaciones y al tamaño de la instancia.

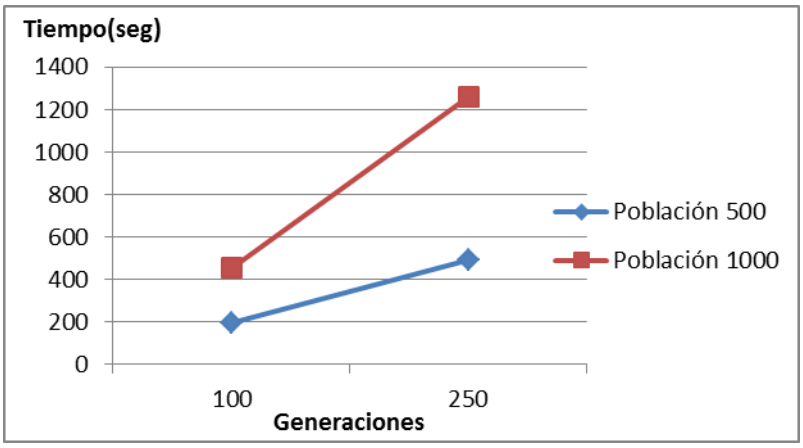


**Figura 26: Tiempo prueba secuencial con instancia 1**

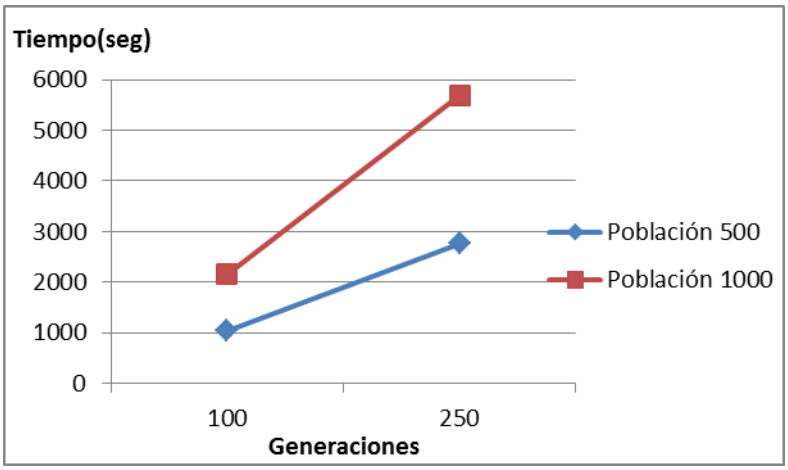


**Figura 27: Tiempo prueba secuencial con instancia 2**

### 3.5 Evaluación de la biblioteca de forma distribuida.

Las pruebas fueron realizadas utilizando la plataforma de tareas distribuidas T-Arenal aprovechando los recursos de cómputo del laboratorio 402 del docente 4 con 30 PC, éstas cuentan con procesador Intel (R) Pentium (R) 4 a 3.00 GHz y memoria principal de 1 GB de RAM, con sistema operativo Linux y una red a 100 Mbps.

En las pruebas distribuidas además de variar el tamaño de la población y el número de generaciones también se varió la topología de migración siguiendo el modelo de islas presentado en el epígrafe 1.6. , utilizando tanto migración en anillo como en estrella.

Como muestra la Figura 28 y la Figura 29, el valor de aptitud obtenido en la instancia 1, mejora al aumentar el número de generaciones y el tamaño de la población, utilizando ambas topologías, aunque se obtienen las mejores soluciones con la topología de estrella.

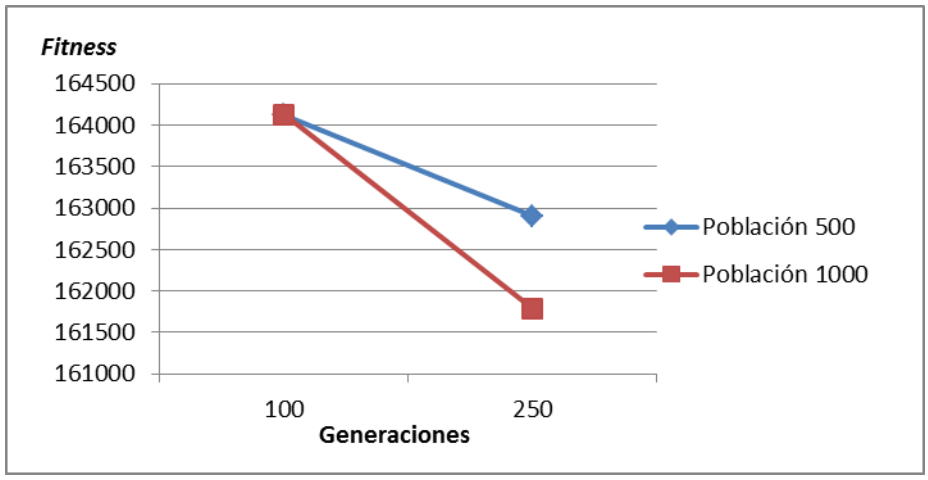


**Figura 28: *Fitness* prueba distribuida con instancia 1 y topología de anillo**

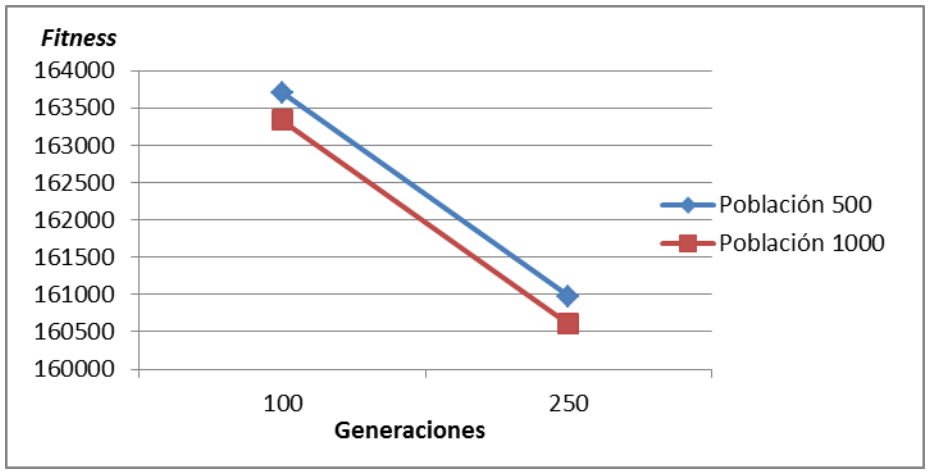


**Figura 29: *Fitness* prueba distribuida con instancia 1 y topología de estrella**

Para la instancia 2, como muestra la Figura 30 y la Figura 31, el comportamiento fue similar y se obtuvieron las mejores soluciones con la topología de estrella cuando la población y el número de generaciones aumentaron.

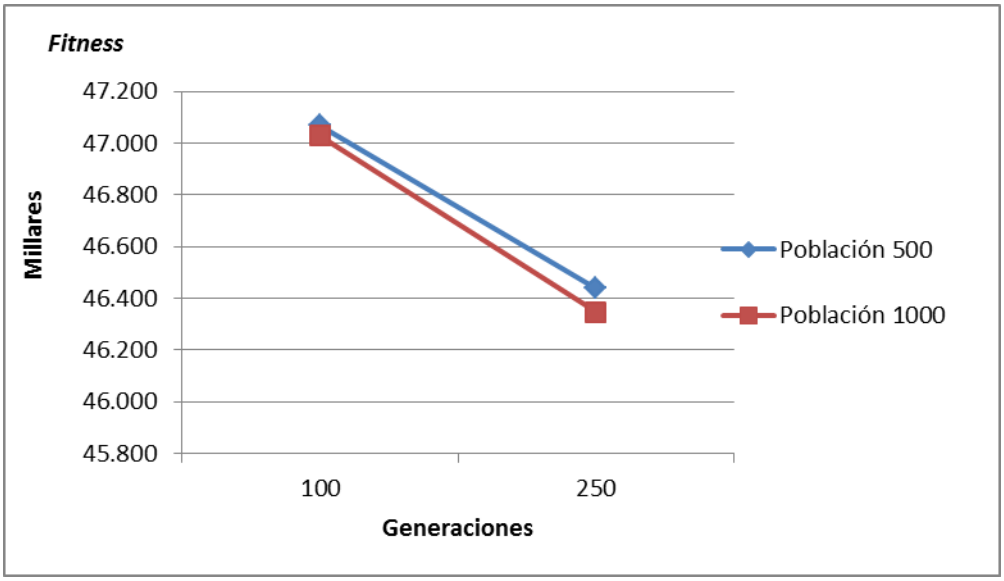


**Figura 30: *Fitness* prueba distribuida con instancia 2 y topología de anillo**

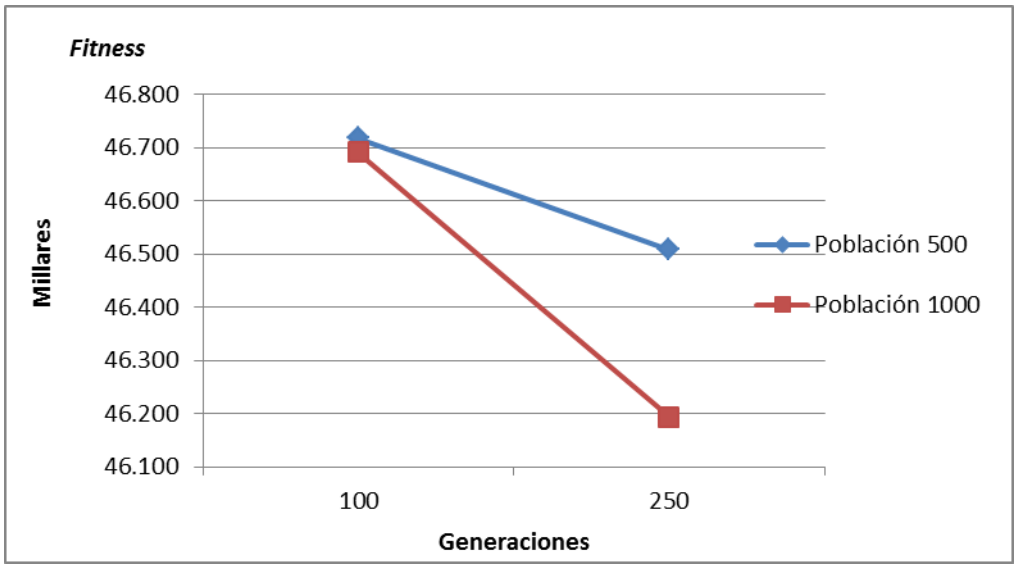


**Figura 31: *Fitness* prueba distribuida con instancia 2 y topología de estrella**

En cuanto al tiempo de ejecución, éste aumentó de forma similar que en las pruebas secuenciales al aumentar el tamaño de la población y el número de generaciones. Como muestra la Figura 32 y la Figura 33, para la instancia 1, se logró el menor tiempo de ejecución con topología de estrella, población de 500 y 100 generaciones, mientras que la topología de anillo presentó el menor tiempo para población de 1000 y 250 generaciones.

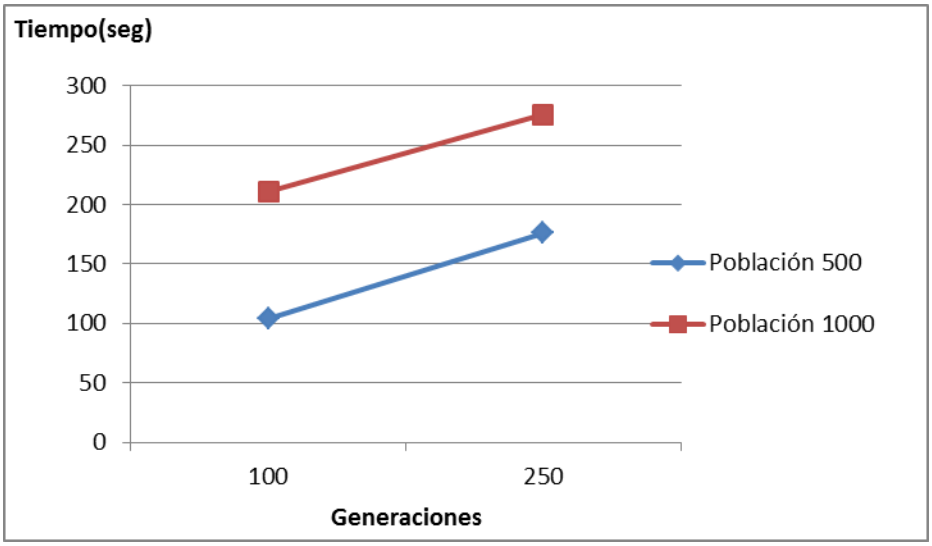


**Figura 32: Tiempo prueba distribuida con instancia 1 y topología de anillo**

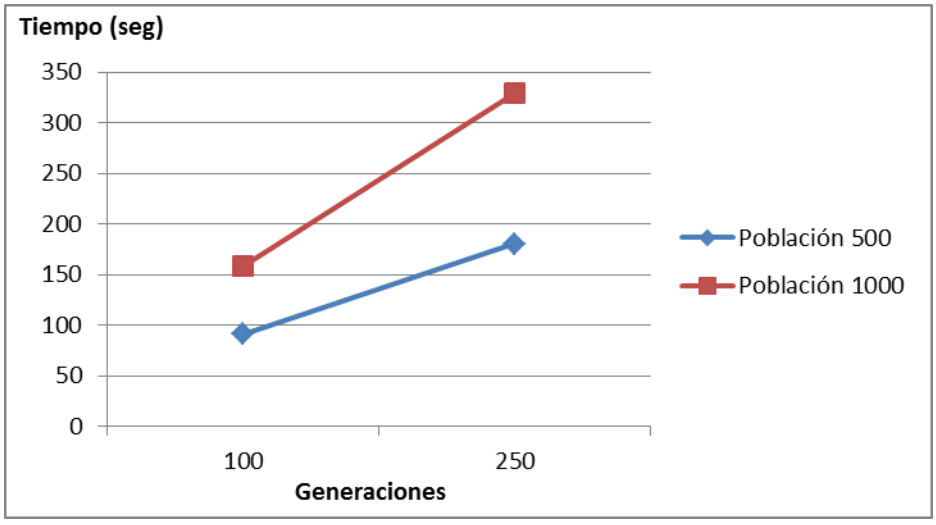


**Figura 33: Tiempo prueba distribuida con instancia 1 y topología de estrella**

Para la instancia 2, como muestra la Figura 34 y la Figura 35, los tiempos mínimos fueron similares con ambas topologías para población de 500 y 100 generaciones, mientras que para población de 1000 y 250 generaciones la topología de anillo presentó menor tiempo.

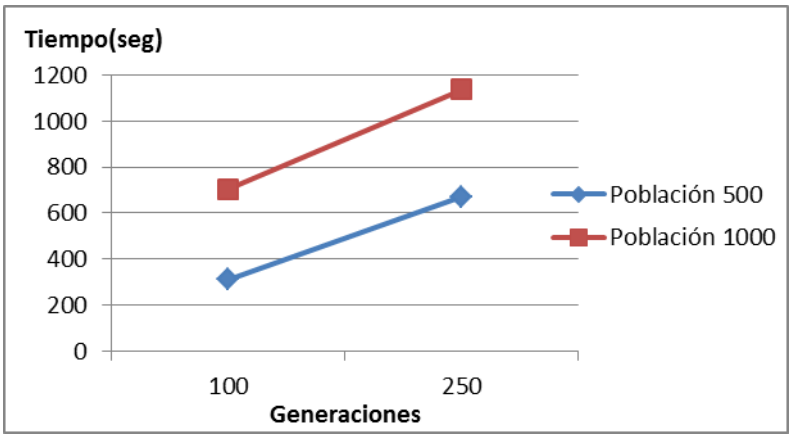


**Figura 34: Tiempo prueba distribuida con instancia 2 y topología de anillo**

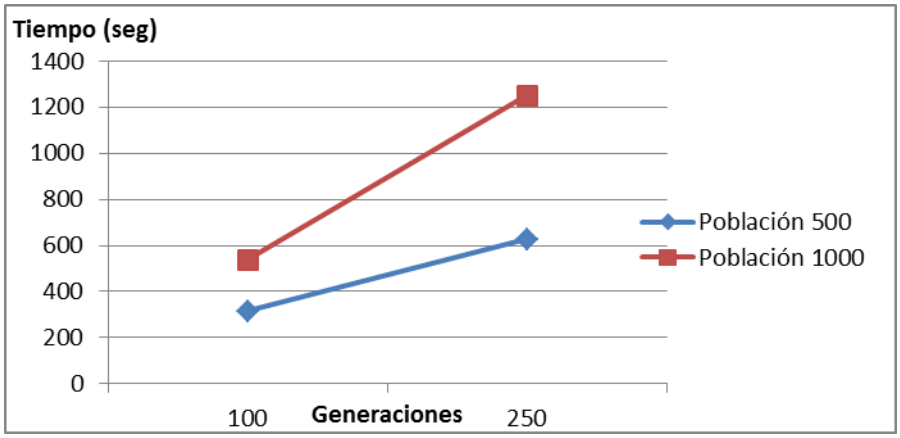


**Figura 35: Tiempo prueba distribuida con instancia 2 y topología de estrella**

### 3.6 Comparación de resultados.

Al analizar los resultados obtenidos en las pruebas secuenciales y distribuidas se puede observar claramente cómo los resultados de las pruebas distribuidas muestran una reducción en el tiempo de ejecución para ambas instancias y además brindan una mejor solución al problema.

La Figura 36 muestra cómo el tiempo de ejecución requerido en ambas instancias es considerablemente menor en la ejecución distribuida que en la secuencial, siendo hasta 4 veces menor para la instancia 2.

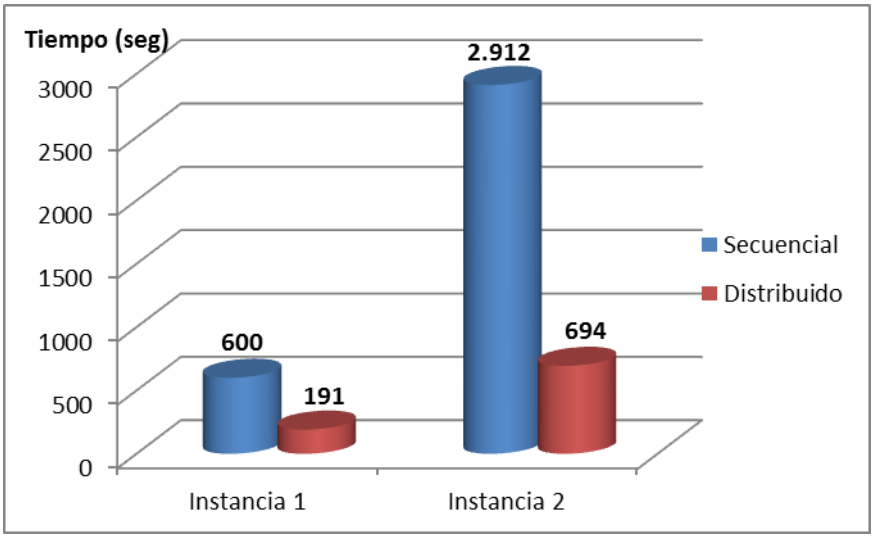


**Figura 36: Comparación de tiempo secuencial y distribuido**

En cuanto a la calidad de la solución, la Figura 37 y la Figura 38 muestran el valor de aptitud obtenido. Se puede observar que el valor alcanzado en la ejecución distribuida es menor que el logrado en la secuencial, en ambas instancias, lo que representa una mejor solución a nuestro problema.

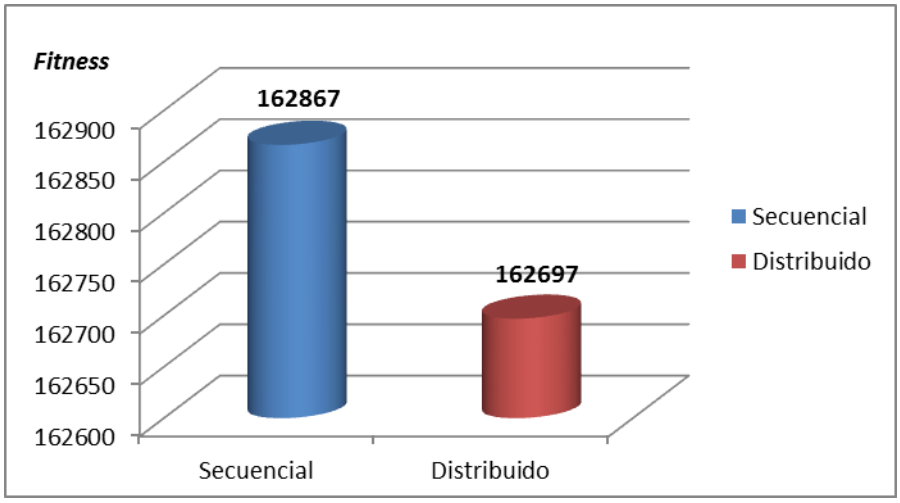


**Figura 37: Comparación de *fitness* secuencial y distribuido con instancia 1**

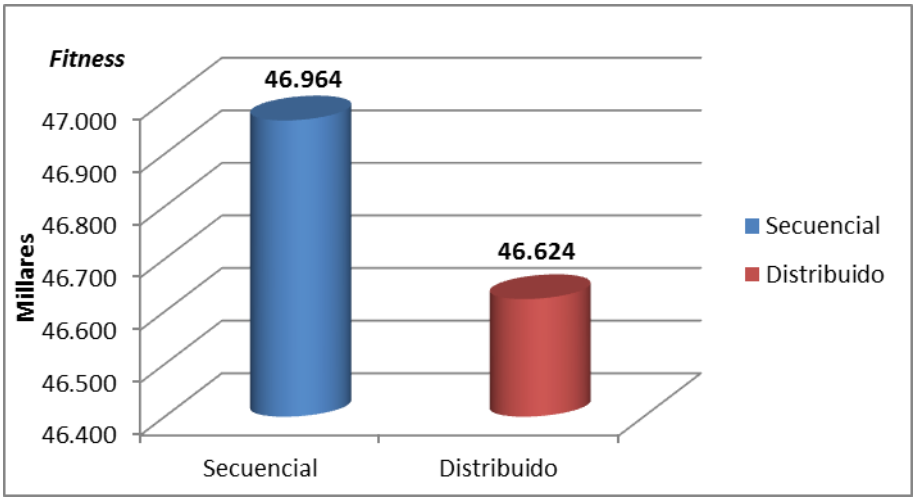


**Figura 38:Comparación de *fitness* secuencial y distribuido con instancia 2**

### 3.7 Métricas de rendimiento.

Para calcular el desempeño logrado de la biblioteca se ha propuesto evaluar dos métricas fundamentales, dichas métricas son:

**Ganancia de velocidad (*Speed-Up*)**

El *Speed-Up* ($SP$), vincula el tiempo de ejecución de un programa secuencial $TS$ y el tiempo total de ejecución de la versión en paralelo $TP$ de dicho programa ejecutado sobre $p$ cantidad procesadores, como se muestra en la ecuación:

$$SP = TS/TP$$

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 ($E$)**

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.

Como se puede observar en la Figura 39 la ganancia de velocidad en función del tamaño del problema alcanzó valores de hasta 5 para la topología de anillo, mientras que para la topología de estrella el valor máximo fue de 4.5.

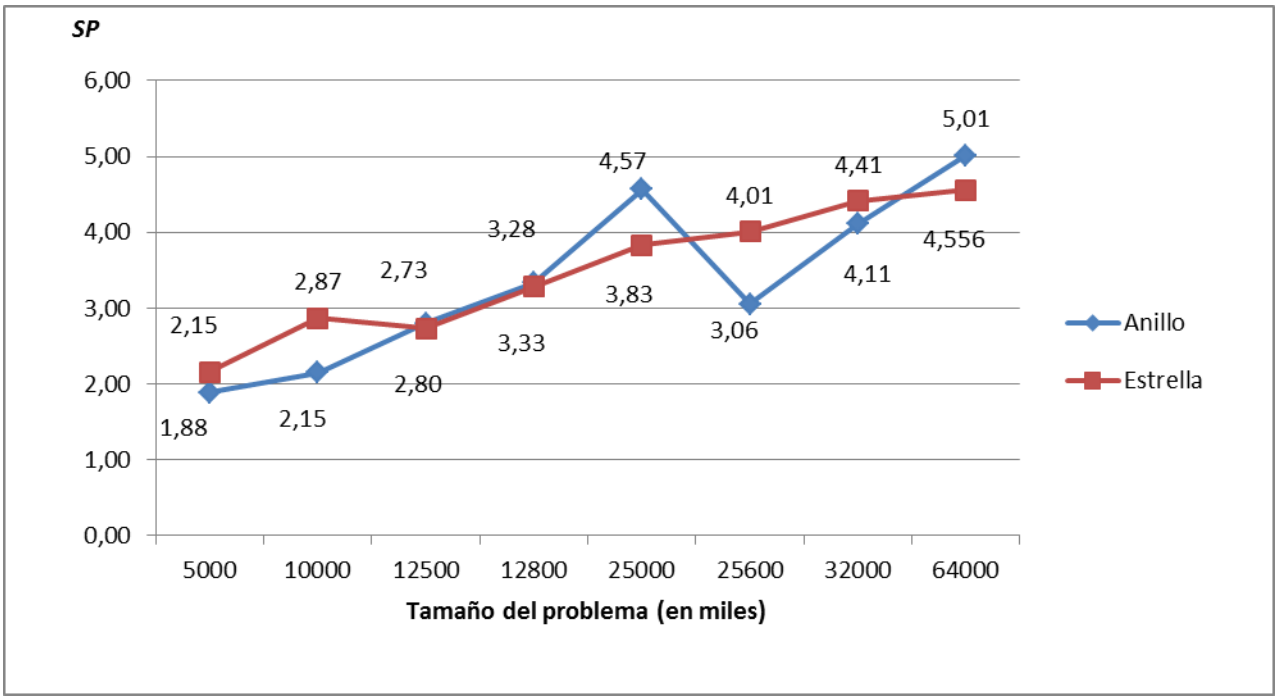


**Figura 39: *Speed-Up* en función del tamaño del problema**

La eficiencia obtenida, como muestra la Figura 40, se mantuvo con una media de 0.24 y con un valor mínimo de 0.15 con la topología de estrella y un valor máximo de 0.36 con la topología de anillo. Al analizar estos resultados se puede observar que la topología de estrella presenta menor eficiencia pero se mantiene más estable en cuanto a ganancia de velocidad y eficiencia, al aumentar el tamaño del problema.

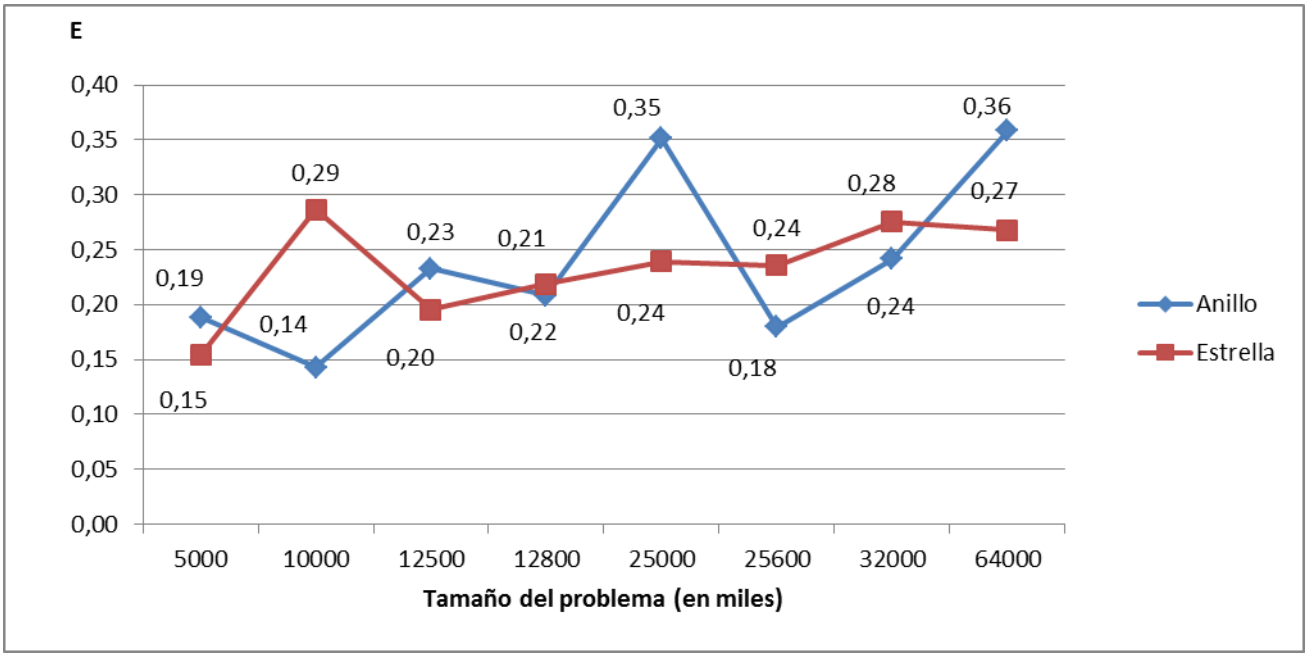


**Figura 40: Eficiencia en función del tamaño del problema**

### 3.8 Valoraciones finales.

Como hipótesis investigativa de este trabajo se planteó que era posible disminuir el tiempo de ejecución y aumentar la calidad de la solución si se empleaban Algoritmos Genéticos sobre la plataforma de cálculo distribuido T-Arenal para resolver problemas de optimización computacionalmente costosos. Como variables se identificaron el tiempo de ejecución y la calidad de la solución.

Los resultados prácticos obtenidos mediante la modelación y resolución del QAP, demostraron que la versión distribuida del algoritmo, empleando el modelo de islas, fue hasta aproximadamente 5 veces más rápido que su versión secuencial cuando se usan hasta 14 PC. Al mismo tiempo el algoritmo distribuido brinda una mejor solución que el secuencial, aunque aumentando el número de generaciones se logra que ambos obtengan resultados similares.

### 3.9 Conclusiones

En este capítulo se demostró la eficiencia de la biblioteca y su desempeño exitoso en el caso de estudio propuesto. Se logró aplicar la biblioteca a un problema de optimización combinatoria, costoso computacionalmente, de forma secuencial y distribuida, alcanzando una reducción considerable de tiempo y obteniendo mejores resultados en las ejecuciones distribuidas.

## Conclusiones generales

La biblioteca brinda facilidad para aplicar Algoritmos Genéticos ya que contiene las clases necesarias para modelar y enfrentar problemas de optimización, computacionalmente costosos, de manera secuencial y distribuida sobre T-Arenal.

En la resolución del caso de estudio se realizaron pruebas de rendimiento y se demostró que el modelo de islas utilizado logra valores de ganancia de velocidad y eficiencia buenos en entornos distribuidos.

El desarrollo de la biblioteca demostró que el empleo de Algoritmos Genéticos sobre la plataforma de cálculo distribuido T-Arenal, para resolver problemas de optimización computacionalmente costosos, según las posibilidades de cómputo de la UCI, disminuye considerablemente el tiempo requerido y aumenta la calidad de la solución.

## Recomendaciones

En esta investigación quedaron abiertos algunos aspectos que merecen ser tratados pues servirían de complemento y continuidad al trabajo realizado, en tal sentido se propone:

- ✓ Implementar otras variantes de selección, cruzamiento y mutación que no estén incluidas en la biblioteca.
- ✓ Estudiar otros modelos de distribución y su posible implementación sobre T-Arenal.

## Bibliografía

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

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

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

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

*"DREAM Distributed Resources Evolutionary Algorithm Machine".* Paechter, B., Baech, T. y Merelo, J.J. 2000. 2000. Proceedings of the. págs. 951-958.

*"JEO: Java Evolving Objects".* García Arenas, Maribel y Merelo, J.J. 2002. New York : s.n., 2002. Proceedings of the Genetic and Evolutionary Computation. pág. 991.

*A discrete binary version of the particle swarm algorithm.* Kennedy, J y Eberhart, R C. 1997. 1997. Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics .

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

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

A. Puder, K. ROMER, y F. PILHOFER. 2006. *Distributed Systems Architecture. A Middleware Approach.* San Francisco : Morgan Kaufmann publishers is an imprint of Elsevier, 2006.

*Adaptation in Natural and Artificial Systems.* Holland, J. H. 1975. 1975.

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

Aguilera, César. *Introducción a los Algoritmos Genéticos.* Granada : Universidad de Granada.

Alba, E y Toya, J.M. 2001. *Analyzing Synchronous and Asyncrhonous Parallel Distributed Genetic Algorithms. Future Generation Computer Systems.* 2001. págs. 451-465.

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

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

Álvarez, Juan Andrés, TRUJILLO M., HENRY y HURTADO G., SANDRA VICTORIA. 2010. *Algoritmos Genéticos.* s.l. : Publicaciones Icesi, 2010.

*An Analysis of Publications on Particle Swarm.* Poli, Riccardo. 2007. 2007.

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

*Ant colony optimization theory: A survey.* Dorigo, M. y Blum, C. . 2005 : s.n.

Bäck, T. 1966. *Evolutionary Algorithms in Theory and Practice.* New York : Oxford University Press, 1966.

Barán, Ing. Benjamín. *Algoritmos Bio-Inspirados.* Paraguay : Universidad Nacional de Asunción Paraguay.

Bellman, R. 1957. *Dynamic Programming.* 1957.

Blum, C y Roli, A. 2003. *Metaheuristics in Combinatorial Optimization:Overview and Conceptual Comparison.* s.l. : ACM Computing Surveys, 2003.

Burkard, R.E, Cela, E y Karisch, S.E. 2002. QAPLib Problems Instances and Solutions. [En línea] 2 de 2002. [Citado el: 20 de 4 de 2012.] http://www.opt.math.tu-graz.ac.at/qaplib/inst.html.

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

Coello, Carlos A Coello. 1995. Red Científica. *Introducción a los Algoritmos Genéticos.* [En línea] 1 de 1995. [Citado el: 16 de 10 de 2011.] http://www.redcientifica.com/doc/doc199904260011.html.

*Comparing inertia weights and constriction factors in particle swarm optimization.* Eberhart, R.C y Shi, Y. 2000. 2000, Vol. 1.

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

2010. Cubadebate. [En línea] 20 de 10 de 2010. [Citado el: 22 de Noviembre de 2011.] http://www.cubadebate.cu/noticias/2010/10/20/biotecnologia-cubana-contribuye-a-la-expectativa-de-vida/.

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

*Distributed Computing:The Condor Experience.* Thain, Douglas , Tannenbaum, Todd y Livny, Miron. 2004. s.l. : John Wiley & Sons, Ltd, 2004.

Distributed.net. [En línea] http://www.distributed.net.

*Efficient local search with search space smoothing: A case study of the traveling salesman problem.* Gu, J. y Huang, X. 1994. 1994.

*Empirical Study of Particle Swarm Optimization.In Proceedings of the 1999 IEEE Congress on Evolutionary Computation.* Shi, Y. y Eberhart, R.C. 1999. 1999.

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

Flynn, MJ. 1972. *Some Computer Organizations and Their Effectiveness.* s.l. : IEEE Trans Comput, 1972.

Fogel, Lawrence J. 1966. *Artificial Intelligence through Simulated .* New York : s.n., 1966.

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

Glover, F y Kochenberger, G. 2002. *Handbook of Metaheuristics.* s.l. : Kluwer Academic Publishers, 2002.

Glover, F. 1986. *Future paths for integer programming and links to artificial intelligence. Computers and Operation Research.* 1986.

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

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

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

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

Hernández, Jorge Raúl Lu. 2010. *Laboratorio de Inteligencia Artificial 1. Práctica 1- Algoritmos Genéticos.* Guatemala : Universidad de San Carlos de Guatemala, 2010.

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

Increase productivity, comunication, and collaboration using UML visual modeling plataform. [En línea] [Citado el: 10 de 12 de 2010.] http://www.visual-paradigm.com/product/vpuml.

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

Jacas, César Raúl García. *Manual del Usuario.Front-end de la Plataforma de Tareas Distribuidas.Interfaz de Escritorio.T-arenalVersión 2.0.*

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

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

Kennedy, J. y Eberhart, R.C. 2001. *Swarm intelligence.* San Francisco : Morgan Kaufmann Publishers, 2001.

Londoño, Natyhelem Gil. 2006. *Algoritmos Genéticos.* Colombia : Universidad Nacional de Colombia, 2006.

López, Gabriela González. 2006. Programación Distribuida. [En línea] 27 de Septiembre de 2006. [Citado el: 20 de Octubre de 2011.] http://pdistribuida.blogspot.com/.

Luis Giraldo, Yuliana Zapata. 2005. *Herramientas de Desarrollo de Ingeniería de SW para Linux.* 2005.

M., Mladenovic y Hansen, P. 1997. *Variable neighborhood search. Computers and Operations.* 1997.

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

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

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

—. 1997-1999. Java. [En línea] CSIC. Todos los derechos reservados, 1997-1999. [Citado el: 10 de 11 de 2011.] http://www.iec.csic.es/criptonomicon/java/quesjava.html.

Martínez., Luis F. Iribarne. 2003. *Un Modelo de Mediación para el desarrollo de Software Basado en Componentes COTS.* España : s.n., 2003.

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

Merelo, Juan Julian. 1997. *Informática Evolutiva.* Granada, España : s.n., 1997.

Meza, Alejandro Morales. 2011. Introducción a la Plataforma Netbeans. [En línea] 25 de febrero de 2011. [Citado el: 29 de 11 de 2011.] http://www.ammeza.com/category/netbeans/.

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

Milian, Carlos, Cruz, Reyder y Trinchet, Dannier. 2011. *Técnicas de Selección en los Algoritmos Genéticos.* Universidad de Ciencias Informáticas. La Habana, Cuba : s.n., 2011.

Míriam, Zito. Cuba Ahora Revista Informativa. [En línea] [Citado el: 15 de Noviembre de 2011.] http://www.cubahora.cu/index.php?tpl=buscar/ver-not_buscar.tpl.html&newsid_obj_id=1023505.

*Optimization by simulated annealing.* Kirkpatrick, S., Gelatt, C. D y Vecchi., M.P. 1983. 1983.

Otero, María Teresa Iglesias. 2009. *Biología + Matemáticas + Informática = Algoritmos Genéticos.* s.l. : Universidad de Coruña, 2009.

*Outline for a logical theory of adaptive systems.* Holland, J. H. 1962. 1962.

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

Palomar, Isabel. 2010. 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.

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

*Particle swarm optimization.* Kennedy, Jammes y Eberhart, R.C. 1995. 1995.

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

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

*P-complete approximation problems.* Sahni, S. y González, T. 1976. 3, 1976, Journal of the ACM, Vol. 23, págs. 555–565.

Pit, LV. 1995. *Parallel genetic algorithms.* s.l. : Leiden University, 1995.

Pit, LV. 1995. *Parallel genetic algorithms.* Leiden University : s.n., 1995.

Quintero, Luis Vicente Santana. 2006. *Una introducción a la computación evolutiva.* 2006.

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

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

Rechenberg, I. 1973. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien .* Alemania : Frommann–Holzboog, 1973.

Reyes, Rigoberto Leander Salgado. 2009. *Librería paralela de algoritmos evolutivos aplicados a problemas computacionalmente costosos.* La Habana : s.n., 2009.

Sanchez, Maria A. Mendoza. 2004. *Metodologias de desarrollo de Software.* 2004.

Schwefel, H.P. 1981. *Numerical Optimization of Computer Models.* Chichester, UK : s.n., 1981.

*Tabu search: Part I.* Glover, F. 1989. 1989.

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

Tanenbaum, A. y Steen , MV. 2002. *Distributed Systems: Principles and Paradigms.* s.l. : Prentice Hall, 2002.

The Great Internet Mersenne Prime Search. [En línea] http://www.mersenne.org/.

*The particle swarm: Explosion, stability, and convergence in a multidimensional complex space.* Clerc, M. y Kennedy, J. 2002. 2002.

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

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

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

Veloso, Gabriel Alejandro y Arce, Rubén. 2011. Algoritmos Genéticos JGAP. *Aplicaciones Informáticas.* [En línea] 29 de Abril de 2011. [Citado el: 3 de noviembre de 2011.] www.scribd.com/doc/54174090/ApuntesJGAP.

Visual Paradigm International Ltd. Increase productivity, comunication, and collaboration using UML visual modeling plataform. [En línea] [Citado el: 10 de 12 de 2010.] http://www.visual-paradigm.com/product/vpuml.

Wolpert, David. 1996. No Free Lunch Theorems. [En línea] 1996. [Citado el: 12 de 6 de 2012.] http://www.no-free-lunch.org/.

# Anexos

- **Anexo 1:** Pruebas secuenciales

| Instancia | Población | Generaciones | *Fitness* | Tiempo (seg) |
|---|---|---|---|---|
| 100 | 500 | 100 | 165062 | 196 |
| 100 | 1000 | 100 | 163962 | 453 |
| 100 | 500 | 250 | 161252 | 492 |
| 100 | 1000 | 250 | 161190 | 1260 |
| 256 | 500 | 100 | 47.324.706 | 1036 |
| 256 | 1000 | 100 | 47.826.616 | 2154 |
| 256 | 500 | 250 | 46.450.398 | 2761 |
| 256 | 1000 | 250 | 46.252.364 | 5695 |

- **Anexo 2:** Pruebas distribuidas

| Prueba | Instancia | Población | Generaciones | Topología | *Fitness* | Tiempo (seg) |
|---|---|---|---|---|---|---|
| 1 | 100 | 500 | 100 | Anillo | 164120 | 104 |
| | 100 | 500 | 100 | Estrella | 163706 | 91 |
| 2 | 100 | 1000 | 100 | Anillo | 164130 | 211 |
| | 100 | 1000 | 100 | Estrella | 163342 | 158 |
| 3 | 100 | 500 | 250 | Anillo | 162906 | 176 |
| | 100 | 500 | 250 | Estrella | 160972 | 180 |
| 4 | 100 | 1000 | 250 | Anillo | 161790 | 276 |
| | 100 | 1000 | 250 | Estrella | 160610 | 329 |
| 5 | 256 | 500 | 100 | Anillo | 47.067.054 | 311 |
| | 256 | 500 | 100 | Estrella | 46.717.542 | 316 |
| 6 | 256 | 1000 | 100 | Anillo | 47.026.934 | 705 |
| | 256 | 1000 | 100 | Estrella | 46.691.232 | 537 |
| 7 | 256 | 500 | 250 | Anillo | 46.437.580 | 672 |
| | 256 | 500 | 250 | Estrella | 46.508.320 | 626 |
| 8 | 256 | 1000 | 250 | Anillo | 46.347.214 | 1136 |
| | 256 | 1000 | 250 | Estrella | 46.193.938 | 1250 |

## Glosario de términos

**IA** Inteligencia Artificial

**ACO** Del Inglés *Ant Colony Optimization*, Optimización por Colonia de Hormigas.

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

**AE** Algoritmos Evolutivos

**PE** Programación Evolutiva

**EE** Estrategias Evolutivas

**AG** Algoritmos Genéticos

**UCI** Universidad de las Ciencias Informáticas

**SD** Sistema distribuido

**IDE** Entorno integrado de desarrollo

**UML** Lenguaje Unificado de Modelado

**LAN** Del Inglés *Local Área Network*, Red de Área Local

**CPU** Del Inglés *Central Processing Unit*, Unidad Central de Procesamiento

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