INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

LABORATORIO DE INTELIGENCIA ARTIFICIAL

Modelos de metaheurísticas multipoblacionales implementadas en paralelo

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A:

ING. DANIEL VERGARA MONTES

DIRECTORES DE TESIS: DR. SALVADOR GODOY CALDERÓN
DR. RENÉ LUNA GARCÍA

MÉXICO, D.F DICIEMBRE 2015

SIP-14 bis

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 11:00 horas del día 01 del mes de diciembre de 2015 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Modelos de metaheurísticas multipoblacionales implementadas en paralelo"

Presentada por el alumno:

VERGARA	**MONTES**	**DANIEL**
Apellido paterno	Apellido materno	Nombre(s)

Con registro: **B 1 3 0 1 3 2**

aspirante de: **MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Directores de Tesis

Dr. Salvador Godoy Calderón

Dr. René Luna García

Dr. Sergio Suárez Guerra

Dr. Ricardo Barrón Fernández

Dr. Jesús Guillermo Figueroa Nazuno

Dr. Francisco Hiram Calvo Castro

PRESIDENTE DEL COLEGIO DE PROFESORES

Dr. Luis Enrique Villa Vargas

ESTADOS UNIDOS MEXICANOS

INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION
EN COMPUTACION
DIRECCION

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México, D.F. el día 02 del mes de Diciembre del año 2015, el (la) que suscribe Daniel Vergara Montes alumno(a) del Programa de Maestría en Ciencias de la Computación, con número de registro B130132, adscrito al Centro de Investigación en Computación, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de Dr. Salvador Godoy Calderón y Dr. René Luna García y cede los derechos del trabajo titulado Modelos de metaheurísticas multipoblacionales implementadas en paralelo, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del (de la) autor(a) y/o director(es) del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección danyverm@gmail.com. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Daniel Vergara Montes

Nombre y firma del alumno(a)

RESUMEN

Para la realización de este trabajo de investigación, nos enfocamos en proponer diferentes modelos de implementación paralela de metaheurísticas multipoblacionales y poder comparar las ventajas que tienen unas implementaciones con respecto a otras. Los modelos de metaheurísticas multipoblacionales más usados en la actualidad son el modelo distribuido y el modelo celular síncrono y asíncrono, los cuales están implementados utilizando memoria compartida y memoria distribuida como modelos paralelos.

A pesar de que existe un gran número de trabajos relacionados a la paralelización de metaheurísticas multipoblacionales, no existen muchas herramientas que permitan utilizar estas implementaciones e inclusive es mucho más difícil encontrar implementaciones del modelo celular tanto síncrono como asíncrono.

Nuestra propuesta consiste en implementar diferentes modelos de metaheurísticas multipoblacionales en algunos modelos paralelos propuestos y poder comparar los resultados obtenidos y determinar cuáles son los mejores parámetros que permiten obtener soluciones de calidad; así como la creación de una biblioteca que permita utilizar estas implementaciones en trabajos futuros de investigación y desarrollo.

ABSTRACT

For this research, we focus on proposing different parallel implementation of metaheuristics multi-population models and to compare the advantages they have few over other implementations. The most commonly used multi-population metaheuristics models today are the distributed model and the synchronous and asynchronous cellular model, which are implemented in parallel models using shared memory and distributed memory.

Although there are some papers related to multi-population metaheuristics parallelization, there are not many tools that use these implementations and even harder to find implementations of the cellular model both synchronous and asynchronous.

Our proposal is to implement different multi-population metaheuristics models in some proposed parallel models and to compare the results and determine what are the best parameters which can obtain quality solutions as well as the creation of a library that allows to use these implementations in future research and development.

AGRADECIMIENTOS

Agradezco a mis padres y a mi hermano que me han dado todo su apoyo desde el momento que decidí estudiar la maestría en ciencias de la computación hasta el final de esta maestría.

Agradezco a mis amigos por su apoyo moral y técnico para poder concluir este trabajo de tesis.

Agradezco a mis profesores y en especial a mis directores de tesis el Dr. Salvador Godoy Calderón y el Dr. René Luna García que me han dirigido y me han dado las herramientas y los conocimientos para realizar esta tesis de forma exitosa.

Agradezco al Instituto Politécnico Nacional por darme triunfos académicos, deportivos y culturales; por darme la visión y las habilidades para mostrar al mundo lo que México es capaz de hacer, porque como Politécnico estoy al servicio de la Patria.

Agradezco el apoyo brindado por CONACyT ya que gracias a las becas que proporciona fomenta la investigación y el estudio de posgrados.

ÍNDICE

ÍNDICE DE ALGORITMOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

- **AE.** Los a*lgoritmos evolutivos* son métodos inspirados por procesos y mecanismos de la evolución biológica. En ellos se mantiene un conjunto de entidades que representan posibles soluciones, las cuales se mezclan, y compiten entre sí, de tal manera que las más aptas son capaces de prevalecer a lo largo del tiempo, evolucionando hacia mejores soluciones cada vez
- **AG.** Los *algoritmos genéticos* son estrategias adaptativas y técnicas de optimización global que pertenecen a los algoritmos evolutivos.
- **AGc.** Los *algoritmos genéticos celulares* son metaheurísticas donde los individuos de la población está conectados en una malla toroidal y solo se permite recombinarse con individuos cercanos.
- **AGd.** Los *algoritmos genéticos distribuidos* son metaheurísticas donde la población es estructurada en pequeñas poblaciones también llamadas *islas,* aisladas unas de otras.
- **AGP.** Los *algoritmos genéticos paralelos* tienen las mismas características que un algoritmo genético clásico con la diferencia en que se implementan siguiendo modelos paralelos.
- **API.** La *interfaz de programación de aplicaciones*, abreviada como API (*Application Programming Interface*), es el conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
- **Aptitud.** Cualidad que hace que un objeto sea apto, adecuado o acomodado para cierto fin.
- **Benchmark.** Es una técnica utilizada para medir el rendimiento de un sistema o componente del mismo.
- **Bytecode.** Código intermedio más abstracto que el código máquina. Habitualmente es tratado como un archivo binario que contiene un programa ejecutable similar a un módulo objeto, que es un archivo binario producido por el compilador cuyo contenido es el código objeto o código máquina.
- **Ciclo hamiltoniano.** Se llama ciclo hamiltoniano a un ciclo en el que aparecen todos los vértices una única vez, es decir, a lo largo del ciclo pasamos una, y sólo una, vez por cada uno de los vértices.
- **Convergencia.** Dicho de una sucesión: Aproximarse a un límite.
- **CUDA.** Es una arquitectura de cálculo paralelo de NVIDIA que aprovecha la gran potencia de la GPU (unidad de procesamiento gráfico) para proporcionar un incremento extraordinario del rendimiento del sistema.

- **CPU.** La unidad central de procesamiento o unidad de procesamiento central (conocida por las siglas CPU, del inglés: *central processing unit*), es el hardware dentro de una computadora u otros dispositivos programables, que interpreta las instrucciones de un programa informático mediante la realización de las operaciones básicas aritméticas, lógicas y de entrada/salida del sistema.
- **DE.** Algoritmo de evolución diferencial.
- **Deadlock.** El Bloqueo mutuo es el bloqueo permanente de un conjunto de procesos o hilos de ejecución en un sistema concurrente que compiten por recursos del sistema o bien se comunican entre ellos.
- **ED.** El algoritmo de *evolución diferencial* es un método de optimización perteneciente a los algoritmos evolutivos. Mantiene una población de soluciones candidatas, las cuales se recombinan y mutan para producir nuevos individuos los cuales serán elegidos de acuerdo al valor de su función de aptitud. Lo que lo caracteriza es el uso de vectores de prueba, los cuales compiten con los individuos de la población actual a fin de sobrevivir.
- **Exclusión múltiple.** Son algoritmos que se usan en programación concurrente para evitar el uso simultáneo de recursos comunes, como variables globales, por fragmentos de código conocidos como secciones críticas.
- **GIL.** Es el mecanismo utilizado en CPython para impedir que múltiples hilos modifiquen los objetos de Python a la vez en una aplicación multihilo.
- **GPU.** La unidad de procesamiento gráfico o GPU (*Graphics Processing Unit*) es un coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador central.
- **Heurística.**En algunas ciencias, manera de buscar la solución de un problema mediante métodos no rigurosos, como por tanteo, reglas empíricas, etc.
- **Hilos.** Un hilo de ejecución o subproceso es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo.
- **MIMD.** "Múltiples instrucciones, múltiples datos" es una técnica empleada para lograr paralelismo. Las máquinas que usan MIMD tienen un número de procesadores que funcionan de manera asíncrona e independiente. En cualquier momento, cualquier procesador puede ejecutar diferentes instrucciones sobre distintos datos.
- **MPP.** *Procesamiento paralelo masivo* es una forma procesamiento colaborativo donde el mismo programa es ejecutado por dos o más procesadores.

- **Núcleo.** Un procesador multinúcleo es aquel que combina dos o más microprocesadores independientes en un solo paquete, a menudo un solo circuito integrado.
- **OC**. La Optimización Combinatoria es una rama de la optimización. Su dominio se compone de problemas de optimización donde el conjunto de posibles soluciones es discreto o se puede reducir a un conjunto discreto.
- **Open source.** El software de código abierto es aquel distribuido bajo una licencia que permite su uso, modificación y redistribución. Como su nombre lo indica, el requisito principal para que una aplicación sea considerada bajo esta categoría es que el código fuente se encuentre disponible. Esto permite estudiar el funcionamiento del programa y efectuar modificaciones con el fin de mejorarlo y/o adaptarlo a algún propósito específico.
- **PVM.** La Máquina Virtual Paralela (conocida como PVM por sus siglas en inglés de *Parallel Virtual Machine*) es una biblioteca para el cómputo paralelo en un sistema distribuido de computadoras. Está diseñado para permitir que una red de computadoras heterogénea comparta sus recursos de cómputo (como el procesador y la memoria RAM) con el fin de aprovechar esto para disminuir el tiempo de ejecución de un programa al distribuir la carga de trabajo en varias computadoras.
- **SA.** *Simulated annealing* (SA) es un algoritmo de búsqueda metaheurística para problemas de optimización global.
- **Semáforos.** Un semáforo es una variable especial (o tipo abstracto de datos) que constituye el método clásico para restringir o permitir el acceso a recursos en un entorno de multiprocesamiento.
- **SIMD.** "Una instrucción, múltiples datos" consisten en instrucciones que aplican una misma operación sobre un conjunto de datos.
- **SISD.** "Una instrucción, un dato" En un único procesador se ejecuta un sólo flujo de instrucciones, para operar sobre datos almacenados en una única memoria.
- **Toroide.** Superficie de revolución engendrada por una curva cerrada y plana que gira alrededor de una recta fija de su plano y exterior a ella.
- **TSP.** El problema del agente viajero se relaciona con el problema de encontrar un ciclo hamiltoniano en un grafo. El problema es: Dado un grafo ponderado G, encuentre en G un ciclo de Hamilton con longitud mínima.
- **XML.** Especificación para diseñar lenguajes de marcado, que permite definir etiquetas personalizadas para descripción y organización de datos.

Capítulo 1

INTRODUCCIÓN

Los problemas de optimización, en la práctica, son problemas complejos que requieren de mucho tiempo de ejecución para poder encontrar su solución. Estas soluciones pueden ser encontradas utilizando métodos exactos, fuerza fruta o metaheurísticas. Los métodos exactos y de fuerza bruta permiten obtener respuestas exactas pero lamentablemente son imprácticos debido a que el tiempo que tardan en encontrar una solución es extremadamente grande. Como método de solución alternativo a este tipo de problemas, las metaheurísticas encuentran soluciones casi óptimas o en algunos casos óptimas en un tiempo razonable.

Existe una clase de metaheurísticas basadas en poblaciones, las cuales se basan en conjuntos de poblaciones que aplican operadores estocásticos a un conjunto de posibles soluciones llamadas individuos. Las principales ventajas que se tienen al utilizar metaheurísticas multipoblacionales son: mayor diversidad de posibles soluciones dentro del espacio de búsqueda, reducción en el número de generaciones que le tarda al algoritmo converger e inclusive tener la posibilidad de utilizar diferentes metaheurísticas en cada una de las poblaciones.

1.1 Justificación

En la actualidad existen varios trabajos relacionados con metaheurísticas multipoblacionales, pero muy pocos implementan modelos paralelos que permiten reducir el tiempo de ejecución, mejorar la calidad de las soluciones de estas metaheurísticas o simplemente son trabajos teóricos que dan propuestas de posibles implementaciones pero utilizando recursos ideales.

Además de lo anterior, existen muy pocas bibliotecas o herramientas que implementan metaheurísticas multipoblacionales, dejando a los investigadores o a los desarrolladores que estén interesados en el uso de metaheurísticas multipoblacionales, en la necesidad de implementar desde cero estos algoritmos.

El trabajo es proponer e implementar algunos modelos paralelos para diferentes metaheurísticas multipoblacionales. Estas implementaciones se harán desarrollando una biblioteca con diferentes herramientas que permita a investigadores y desarrolladores realizar experimentos utilizando metaheurísticas multipoblacionales en paralelo y poder analizar el comportamiento de las metaheurísticas.

1.2 Objetivos

1.2.1 Objetivo general

Proponer algunos modelos de implementación paralela para las metaheurísticas multipoblacionales más ampliamente usadas e implementarlos en una biblioteca para su uso en la investigación.

1.2.2 Objetivos específicos

- Implementar los modelos propuestos en diferentes metaheurísticas multipoblacionales.
- Seleccionar un conjunto de métricas de evaluación para ser usadas sobre todas las implementaciones.
- Desarrollar una biblioteca para apoyar a usuarios en la implementación de metaheurísticas multipoblacionales en paralelo (Multi-Population Heuristic Studio – MPHStudio).

1.3 Organización del documento

Este documento está dividido principalmente en 6 secciones. En la primera sección se da una introducción al tema que abordará la tesis como son sus objetivos y su justificación. La segunda parte contiene un análisis de las principales investigaciones y estudios que se han desarrollado con respecto al tema de metaheurísticas multipoblacionales. La tercera sección contiene un amplio marco teórico donde se abordan los temas de metaheurísticas, metaheurísticas multipoblacionales y los dos principales modelos. También se toman temas de paralelización, como son sus clasificaciones, principales tecnologías y las métricas más usadas para medir el rendimiento de las implementaciones. En el cuarto capítulo se explica el modelo propuesto, que se basa principalmente en la paralelización del modelo de islas y del modelo celular. Además se explica la biblioteca que se desarrolló para las implementaciones de estos modelos así como una serie de herramientas creadas para el análisis del comportamiento de evolución y resultados de los algoritmos genéticos. El capítulo cinco se realizan una serie de pruebas para medir el rendimiento de las implementaciones y la calidad de las soluciones que los algoritmos implementados obtienen. Para ello se utilizaron algunas funciones de optimización combinatoria y algunas funciones de optimización numérica. Por último se dan las conclusiones de este trabajo así como la mención del trabajo a futuro y mejoras que podrán llevarse a cabo a la biblioteca.

Capítulo 2

ESTADO DEL ARTE

Como punto de partida de este capítulo, hacemos referencia al trabajo de Manda et al. [1] en el cual nos presenta un estudio de las metaheurísticas basadas en población más usadas y estudiadas, así como algunas áreas de aplicación de cada una de ellas. En la siguiente tabla se muestran las metaheurísticas mencionadas así como algunas áreas de aplicación.

Metaheurísticas	Aplicaciones
Algoritmos genéticos	Robótica, Redes neuronales, Redes semánticas, Selección de cartera en gestión financiera
Optimización por colonia de hormigas	Asignación cuadrática, Coloración de grafos, Redes bayesianas, Enrutamiento de paquetes en red
Optimización por cúmulo de partículas	Redes neuronales artificiales, Redes neuronales difusas, Cuantificación de color en imágenes
Evolución diferencial	Optimización de flujos de poder, Imagen electromagnética, Agrupamiento automático, Problemas de optimización dinámica
Colonia artificial de abejas	Problemas de optimización numérica, Problema de reconfiguración de red
Optimización basada en enseñanza y aprendizaje	Localización de reguladores automáticos de voltaje en sistemas distribuidos, Agrupamiento de datos

Tabla 1 - Tabla de metaheurísticas basadas en población y algunas aplicaciones

El trabajo de Alba et al. [2] es una recopilación de las tecnologías más actuales que se están manejando en el ámbito de las metaheurísticas, los modelos paralelos que se están trabajando en la actualidad, las nuevas aplicaciones que van teniendo así como algunas herramientas que se han desarrollado con diferentes modelos, lenguajes y tecnologías.

Las tecnologías más utilizadas en los últimos tiempos son: multinúcleo, unidades de procesamiento gráfico (GPUs), combinaciones de GPU y CPU y computación en la nube. Cada una de estas tecnologías cuenta con un conjunto de bibliotecas y lenguajes de programación que permiten poder explotar cada una de ellas. Como ejemplo tenemos a las bibliotecas MPI que nos permiten sacar ventaja de implementaciones basadas en memoria distribuida; las bibliotecas de CUDA desarrolladas por NVIDIA que permiten utilizar los GPUs de las tarjetas de video fabricadas con su tecnología y por último los método embebidos en los lenguajes de programación que nos permiten crear procesos, hilos y comunicar un equipo con otro.

Con una descripción más detallada de estas herramientas podemos dividirlas en dos tipo: las herramientas de memoria compartida y las herramientas de memoria distribuida.

Las herramientas de memoria compartida más comunes son:

Hilos

Los hilos son objetos dinámicos cada uno de los cuales ejecuta una secuencia de instrucciones y que comparten entre sí los recursos del proceso [3].

Los procesos multihilos son en sí programas concurrentes, los cuales brindan ciertas ventajas sobre procesos múltiples: intercambio mucho más rápido de contexto entre los hilos, bajo uso de recursos, comunicación simultánea y algunas aplicaciones paralelas se ajustan bien al modelo de hilos [4].

OpenMP

OpenMP es un conjunto de directivas de compilación, bibliotecas de rutinas y variables de ambiente, que proveen un modelo de programación paralela para la arquitectura de memoria compartida.
Las directivas, biblioteca y variables de entorno, permiten al usuario crear y manejar programas en paralelo portables. Estas directivas abarcan lenguajes de programación como C, C++ y Fortran.

La API de OpenMP se encarga de la paralelización, mientras que el programador explícitamente especifica las acciones a tomar por el compilador [5].

En cuanto a las herramientas de memoria distribuida se encuentran:

Sockets

Los sockets son un conjunto de estructuras de datos y funciones de C que permiten al programador establecer canales entre dos computadoras para implementar aplicaciones distribuidas.

Las ventajas de utilizar la API de sockets son: alta estandarización y un completo control de las primitivas de comunicación.

PVM

PVM (*Parallel Virtual Machine*) es un software que permite la utilización de una red heterogénea de computadoras paralelas y seriales como un único sistema.
Las ventajas de un sistema PVM son la amplia aceptabilidad y sus facilidades computacionales heterogéneas, incluyendo la tolerancia a fallos y la interoperabilidad.

MPI

MPI (*Message-Passing Interface*) es una biblioteca de funciones de paso de mensajes que su idea principal es que múltiples procesos de MPI corren en diferentes procesadores, y es estos procesos se comunican mediante el envío de mensajes a través de una infraestructura provista por MPI.

Los procesos en un programa distribuido son escritos en un lenguaje secuencial(C, Fortran), y ellos se comunican y sincronizan mediante la llamada a funciones de la biblioteca de MPI.

Los programas que utilizan las bibliotecas de MPI siguen un estilo SPMD (*Single program, multiple data*), esto quiere decir que cada proceso ejecuta una copia del mismo programa. Cada instancia del programa puede determinar su identificador y hacer las acciones que le correspondan.

MPI está compuesto por un conjunto de 128 funciones que permiten la comunicación proceso a proceso, comunicación grupal, ajuste y control de los grupos de comunicación y la interacción con el entorno de ejecución.

En los trabajos de Long et al. [6], se dice que el diseño de un adecuado modelo de hilos, donde se tenga en cuenta la organización de los hilos, la comunicación y la compartición de datos entre los hilos puede mejorar el rendimiento significativamente. En un modelo distribuido (islas), una isla puede ser implementada en un hilo sin importar que la comunicación entre hilos sea costosa, ya que la comunicación solo se realiza durante la migración. Siguiendo este modelo, entre más hilos se tenga, mayor número de islas habrá y la calidad de la solución será menor. En sus experimentos, hacen la comparación entre el modelo Maestro-Esclavo, modelo síncrono de islas, y modelo asíncronos de islas. En sus resultados observaron que los modelos de islas son mucho más rápidos que el modelo maestro-esclavo, pero de forma contraria cuando el número de hilos aumentaba, la calidad de las soluciones en los modelos de islas era menor que el modelo maestro-esclavo. En otro

experimento realizado, los modelos de islas tardaron menos tiempo en alcanzar la solución óptima. Con respecto a estos experimentos, llegaron a la conclusión de que el modelo asíncrono de islas puede obtener una buena solución en un tiempo corto con un tamaño de población grande en una arquitectura multinúcleo.

En [7] Sena et al. implementaron un algoritmo genético utilizando un paradigma Maestro-Esclavo sobre un modelo distribuido. Cada *esclavo* crea su población aleatoriamente y ejecuta el algoritmo genético, cuando termina una generación regresa el individuo óptimo al *maestro*. La función del maestro es sincronizar y pasar los parámetros a cada individuo. Utilizando el modelo distribuido cada esclavo es una isla y en determinado momento los individuos más aptos migran a otras islas. La topología de conexión entre las islas, es una topología de anillo. Para esta implementación utilizaron el conjunto de bibliotecas llamadas *Parallel Virtual Machine(PVM).*
Con este trabajo concluyeron que al incrementar el número de esclavos arbitrariamente reduce el rendimiento del algoritmo; además con sus experimentos observaron que la variación en la frecuencia de migración afecta notablemente la velocidad de convergencia en su implementación.

Hablando de los efectos que tiene la variación de la frecuencia de migración en el modelo de islas, en el trabajo de Rucinski et al. [8] se realizó un análisis del impacto de la topología de migración en el rendimiento del modelo de islas. Ellos utilizaron el algoritmo paralelo de evolución diferencial (ED) en inglés llamado *Differential Evolution* (DE) y el algoritmo paralelo de recocido simultaneo en inglés llamado *Simulated Annealing* (SA).
Los resultados que obtuvieron y las conclusiones a las cuales llegaron, utilizando como conjunto de pruebas problemas de optimización global, fueron que la topología de migración afecta la calidad de la solución obtenida y el tiempo de convergencia. Este impacto suele ser más evidente en grandes redes de computadoras así como en arquitecturas multi-núcleo y cómputo distribuido.
Las pruebas permitieron concluir que cierto tipo de topologías deben ser evitadas en ambas metaheurísticas, mientras que otras con buenas para una y malas para otra. Para el algoritmo DE, es mejor la topología de anillo y por el contrario es mejor evitar la topología todos conectados. En cuanto al algoritmo SA, una buena topología es el hipercubo, mientras que la topología de anillo y cadena deben ser evitadas.

En el trabajo de Andalon-García et al. [9] implementaron tres topologías de conexión entre islas en un modelo distribuido; las topologías fueron: estrella, anillo unidireccional y anillo bidireccional. Su

implementación la realizaron utilizando la biblioteca llamada *Message Passing Interface (MPI)* y para sus pruebas utilizaron una serie de funciones benchmark.
Con base en sus pruebas, el mejor tiempo obtenido fue con la topología de estrella, además la aceleración utilizando esta topología resultó ser mejor que las otras dos. En cuanto a la calidad de la solución, con las tres topologías utilizadas, resultaron semejantes los resultados, esto debido, según ellos, a la baja cantidad de ejecuciones que realizaron.

El trabajo de Baños et al. [10] consistió en el análisis de rendimiento y calidad de la solución de ciertos paradigmas de paralelización de metaheurísticas basadas en población, en específico el algoritmo de recocido simultaneo de Pareto conocido en inglés como *Pareto Simulated Annealing* (PSA). Los paradigmas de paralelización utilizados fueron: Maestro-Esclavo, modelo de islas, multi-arranque y modelos de islas con espacio de búsqueda dividido. El problema que utilizaron para sus pruebas fue el problema de la partición de un grafo.
Los resultados que obtuvieron con base en sus experimentos fueron: En términos de la calidad de las soluciones, el modelo de multi-inicio no es adecuado para la metaheurística utilizada, ya que impacta negativamente. El modelo Maestro-Esclavo obtiene peores soluciones que el modelo de islas. El modelo de islas con espacio de búsqueda dividido, obtuvo los mejores resultados de los cuatro paradigmas. En cuanto al tiempo de ejecución, el modelo de islas obtuvo la mejor aceleración y la peor aceleración fue del modelo Maestro-Esclavo, principalmente cuando el tamaño de la población incrementa.

En el trabajo de Alba et al. [11], se realizó un estudio del impacto que tiene la sincronización en las implementaciones de algoritmos genéticos paralelos. En un modelo de islas, si los procesadores que se ocupan son de diferente tipo, habrá un comportamiento diferente del algoritmo entre cada una de las islas. Usando el problema de la esfera con 16 variables, su modelo asíncrono, reduce el tiempo de búsqueda con respecto al modelo síncrono.

En [12] Alba et al. comparan algoritmos genéticos celulares síncronos y asíncronos en problemas discretos y continuos. Sus resultados muestran, con respecto a los problemas discretos, los algoritmos asíncronos son más eficientes, pero los algoritmos síncronos obtienen resultados con mejor aptitud. Por el contrario, en problemas continuos, los algoritmos asíncronos son mejores en calidad de solución, mientras que los síncronos son mejores en eficiencia.

En el 2010 Pinel et al. [13] proponen un algoritmo genético celular asíncrono para procesadores multinúcleo. En esta propuesta, dividen la población en un número de bloques continuos con un mismo tamaño de individuos. Cada bloque se le asigna a un hilo diferente, el cual evolucionará los individuos del bloque. En sus resultados observaron que un bloque causa retrasos en la sincronización y empeora el rendimiento cuando el número de hilos aumenta.

En [14] De Andrés designa un capítulo para hacer referencia a algunos autores en cuanto a los parámetros óptimos de un algoritmo genético clásico. Con base a los experimentos que cita, llega a la conclusión de que el uso de tamaños grandes de población mayores a 200 individuos con probabilidades altas de mutación no mejora el algoritmo genético, así como poblaciones muy pequeñas con baja probabilidad de mutación. En cuanto al operador de cruza, el mejor la cruza de dos puntos que el de un punto.

Pinho et al. [15] proponen un biblioteca escrita en Java que ofrece un conjunto de metaheurísticas implementadas en dos modelos de paralelización: modelo de islas y paralelización de la función de evaluación de la aptitud. Los ambientes paralelos que utilizan son: multinúcleo, cluster y malla. Para la evaluación de su biblioteca utilizaron dos problemas de procesos biológicos. Con su implementación del modelo de islas, al reducir la frecuencia de migración aumentó la aceleración pero la calidad de la solución disminuyó. Los mejores resultados en cuanto a la aceleración fue con 2,4 y 8 islas.

En la siguiente tabla se muestran algunas bibliotecas que implementan metaheurísticas multipoblacionales. Esta tabla muestra las principales características así como sus ventajas y desventajas de cada una de ellas.

Biblioteca	Características	Ventajas	Desventajas
Global Optimization toolbox for matlab [16]	Este toolbox proporciona métodos para la búsqueda de soluciones globales a problemas que contienen máximos y mínimos. Incluye búsqueda global, búsqueda de patrones, algoritmos genéticos y Simulated Annealing	• Modelo de islas • Herramientas de monitoreo en el proceso de solución • Interfaz gráfica para ejecución de pruebas. • Herramientas de análisis de resultados.	• Licencia Comercial • Se requiere comprar matlab y luego el toolbox • La topología de conexión entre las poblaciones únicamente es el de anillo. • Se requiere el toolbox *Parallel Computing* para

		• Algoritmo genético multiobjetivo.	paralelizar las ejecuciones.
Mallba [17]	Biblioteca de algoritmos de optimización combinatoria escrita en C++ y utiliza MPI como tecnología de paralelización. Cuenta con algoritmos genéticos, cúmulo de partículas, evolución diferencial entre otras.	• Versiones celulares y distribuidas. • Implementación paralela utilizando MPI	• Última actualización en el 2006. • Compilación en cada implementación. • Instalación complicada debido al uso de bibliotecas viejas.
Inspyred [18]	Framework para la creación de algoritmos bioinspirados escrito en python. Tiene implementados los siguientes algoritmos: Algoritmos genéticos, colonia de hormigas, cúmulo de partículas, evolución diferencial entre otros. Utiliza el módulo de multiprocesamiento de python como tecnología de paralelización.	• Modelo de islas. • Implementación multiplataforma. • Herramientas de análisis de resultados. • Código abierto	
Watchmaker [19]	Es un framework de alto rendimiento escrita en Java. Cuenta con la implementación del algoritmo genético y estrategias evolutivas.	• Modelo de islas. • Multiplataforma. • Interfaz gráfica para pruebas. • Procesamiento distribuido • Multihilo como tecnología de paralelización.	•Compilación en cada implementación.
PyGMO [20]	Es una biblioteca científica que contiene un gran número de problemas y algoritmos de optimización en paralelo utilizando el modelo de islas.	• Modelo de islas. • Una gran variedad de algoritmos. • Gran variedad de problemas implementados.	•Uso de un gran número de bibliotecas externas. •Difícil instalación.

		• Herramientas de análisis de resultados.	

Tabla 2 - Tabla comparativa de bibliotecas que implementan metaheurísticas multipoblacionales

Las bibliotecas Mallba, Inspyred y Watchmaker a pesar de que en su documentación se menciona que manejan el modelo distribuido, solamente están preparadas para trabajar de manera paralela con ese modelo, pero es necesario implementar un modelo en donde se establezca la política de migración a ocupar.

En cuanto a bibliotecas con modelos celulares implementados, la única biblioteca con que se puede trabajar estos modelo es la biblioteca Mallba, la cual está preparada para trabajar con metaheurísticas en paralelo, pero es necesario desarrollar el modelo celular, es decir organizar la población en malla, la forma y tamaño del vecindario y el método de reemplazo de los individuos en la población ya sea siguiendo el modelo celular síncrono o asíncrono.

Capítulo 3

MARCO TEÓRICO

3.1 Introducción

En la optimización tratamos de encontrar las mejores alternativas de solución a un problema dado, el cual puede ser modelado por medio de un conjunto de variables de decisión con sus dominios y restricciones, estos se dividen en tres categorías:

i) Exclusivamente variables discretas (El dominio de cada variable consiste de un conjunto finito de valores discretos).
ii) Exclusivamente variables continuas (Dominio de variable continua).
iii) Variables discretas y continuas

Un problema de optimización se define como una tupla (S, f)

donde

- $S \neq 0$ representa el espacio de búsqueda.
- $f: S \rightarrow \mathbb{R}$ es la función objetivo.

El objetivo consiste en encontrar un conjunto de valores adecuados de forma que la solución representada por estos valores $i^* \in S$ satisfaga la siguiente desigualdad:

$$f(i^*) \leq f(i), \qquad \forall\, i \in S$$

Se puede establecer una igualdad entre tipos de problemas de maximización y minimización de la siguiente forma.

$$max\{f(i)|i \in S\} \equiv min\{-f(i)|i \in S\}$$

Las metaheurísticas fueron originalmente desarrolladas para tratar problemas de variables discretas también llamados problemas de optimización combinatoria (OC).

Muchos algoritmos han sido desarrollados para tratar estos problemas los cuales pueden ser clasificados como algoritmos completos o algoritmos de aproximación. Los algoritmos completos, garantizan encontrar, para cada instancia de un problema OC de tamaño finito, una solución óptima en un tiempo adecuado. Para los problemas OC que son NP-Completos [21], no existe un tiempo polinomial, asumiendo que $P \neq NP$. Es por eso que los métodos completos necesitan tiempo computacional exponencial en el peor de los casos, lo que conduce a un tiempo computacional muy alto para fines prácticos. Es por ello que los métodos de aproximación para la solución de problemas OC han recibido mucha atención en los últimos 40 años. En los métodos de aproximación, se sacrifica la garantía de encontrar una solución óptima pero se pueden obtener buenas soluciones en un tiempo significativamente menor.

Entre los métodos de aproximación se dividen comúnmente en heurísticas constructivas y métodos de búsqueda local.

3.1.1 Heurísticas constructivas

Las heurísticas constructivas son típicamente los métodos de aproximación más rápidos. Ellas se orientan a la obtención de una solución a partir del análisis y selección paulatina de las componentes que la forman. Esto se realiza hasta que una solución es completada o con base en un criterio de paro.

3.1.2 Métodos de búsqueda local

Los algoritmos de búsqueda local empiezan con alguna solución inicial e iterativamente tratan de remplazar la solución actual por una mejor dentro de un vecindario definido apropiadamente.

De acuerdo a Blum, Roli y Alba [22] Un vecindario es definido formalmente como:

Definición 1: Una estructura de vecindario es una función $N: S \rightarrow 2^S$ que asigna a cada $s \in S$ un conjunto de vecinos $N(s) \subseteq S$. *N(s)* es llamado el vecindario de *s*. La aplicación de dicho operador que produce un vecino s' $\in$ *N(s)* de una solución *s* es comúnmente llamado un **movimiento**.

Una solución $s^* \in S$ es llamada una solución mínima global (o mínimo global) si para toda $s \in S$ cumple que $f(s^*) \leq f(s)$. El conjunto de todas las soluciones mínimas globales se denota por S^*.

Adicionalmente debemos definir el concepto de solución mínima local.

Definición 2: Una **Solución mínima local (o mínimo local)** con respecto a una estructura de vecindario *N,* es una solución $\hat{s}$ tal que $\forall s \in \mathcal{N}(\hat{s}): f(\hat{s}) \leq f(s)$. Llamamos $\hat{s}$ una solución mínima global estricta si $\forall s \in \mathcal{N}(\hat{s}): f(\hat{s}) < f(s)$.

3.1.3 Metaheurísticas

Durante los años 70s, apareció un nuevo tipo de algoritmo que básicamente trata de combinar heurísticas básicas para explorar un espacio de búsqueda más efectiva y eficientemente. Estos métodos son llamados metaheurísticas el cual se deriva de dos palabras griegas. *Heurística* deriva del verbo *heuriskein* que significa "encontrar", mientras que el sufijo *meta* significa "más allá, en un nivel superior".

Con las diferentes definiciones de metaheurísticas encontradas en la literatura podemos obtener ciertas propiedades y características:

- Las metaheurísticas son estrategias que "guían" el proceso de búsqueda.
- El objetivo es mejorar la exploración del espacio de búsqueda con el fin de encontrar (o aproximarse) a soluciones óptimas.
- Las metaheurísticas deben de incorporar mecanismos para evitar caer en limitadas áreas del espacio de búsqueda.
- En la actualidad muchas metaheurísticas avanzadas usan la experiencia para la búsqueda para guiar la búsqueda.

En resumen, las metaheurísticas son estrategias de alto nivel para la exploración del espacio de búsqueda mediante el uso de diferentes métodos. Es de gran importancia que exista un balance entre la diversificación y la intensificación. El término de diversificación generalmente se refiere a la exploración del espacio de búsqueda, mientras que el término de intensificación se refiere a la explotación de la experiencia acumulada de búsqueda. Este balance es importante ya que por un lado se requiere la identificación rápida de regiones en el espacio de búsqueda con soluciones de gran calidad y por otro lado no desperdiciar mucho tiempo en regiones del espacio de búsqueda que ya hayan sido exploradas o que no proveen soluciones de alta calidad.

El objetivo que las metaheurísticas es escapar del mínimo local con el fin de proceder en la exploración del espacio de búsqueda y moverse con la esperanza de encontrar un mejor mínimo local.

Existen diferentes formas de clasificar y describir algoritmos de metaheurísticas, dependiendo de las características seleccionadas para diferenciar unos de otros tenemos [23]:

- Inspiradas en la naturaleza vs No inspiradas en la naturaleza
- Función objetivo dinámica vs Función objetivo estática.
- Simple punto(basadas en trayectoria) vs Basadas en poblaciones

3.1.4 Métodos basados en población

Los métodos basados en población, manejan en cada iteración del algoritmo un conjunto de soluciones llamada población.

El rendimiento depende fuertemente en la forma en que la población es manipulada. En el cómputo evolutivo, la población de individuos es modificado por operaciones de recombinación y mutación.

3.1.5 Metaheurísticas descentralizadas

En las implementaciones paralelas es muy común el uso de estructuras de poblaciones. Las más conocidas son las estructuras **distribuidas** [24] y las **celulares** [25].

Una estructura distribuida es un modelo multipoblacional (isla) que realiza intercambio de individuos entre las islas que lo componen.

Una política de migración controla el tipo de distribución que será usada. Esta política debe definir la **topología** de la isla, cuando ocurre la migración, que individuos serán intercambiados, la sincronización entre las sub-poblaciones y el tipo de integración de los individuos intercambiados entre las sub-poblaciones objetivo.

Las estructuras celulares son modelos basados en vecindarios. En este modelo un individuo dado tiene su propio grupo de parejas potenciales definido por un vecindario de individuos; al mismo tiempo un individuo pertenece a otros grupos.

3.2 Arquitectura de computadoras paralelas

Un modelo muy popular para las computadoras paralelas es el modelo introducido por Michael Flynn a mediados de 1960 [26] (Ver Tabla 3).

El modelo de Flynn está basado en la idea de instrucciones y el flujo de datos. Hay cuatro posibles combinaciones, convencionalmente llamado SISD por sus siglas en ingles *Simple Instruction, Single Data*, SIMD (*Single Instruction, Multiple Data),* MISD (*Multiple Instruction, Single Data)* y MIMD (*Multiple Instruction, Multiple Data)*. Este modelo se ilustra en la siguiente figura.

	Una instrucción	Múltiples instrucciones
Un dato	SISD	MISD
Múltiples datos	SIMD	MIMD

Tabla 3 - Arquitectura Flynn

- La arquitectura SISD corresponde la clásica computadora personal con un solo procesador.

Figura 1 - Sistema SISD

- La arquitectura SIMD, la misma instrucción es ejecutada por todos los procesadores en cada ciclo de reloj sobre diferentes datos. Una computadora con esta arquitectura es compuesta por cientos o incluso miles de procesadores cada uno con una pequeña localidad de memoria.

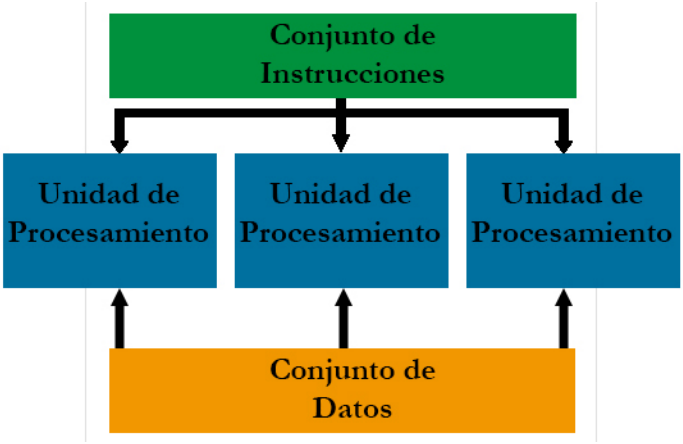


Figura 2 - Sistema SIMD

- En el tipo MISD, cada una de las unidades de procesamiento en paralelo ejecutan operaciones independientemente unas de otras sobre un mismo conjunto de datos. Este tipo de computadoras son difíciles de encontrar en la práctica.

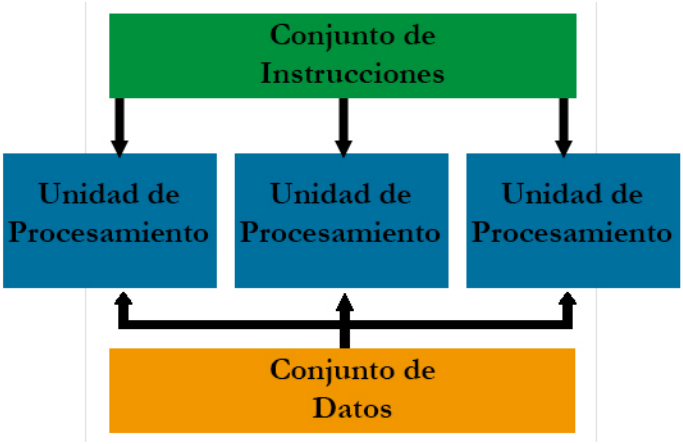


Figura 3 - Sistema MISD

- Por último en el tipo MIMD, diferentes datos y diferentes conjuntos de instrucciones pueden ser cargados en diferentes procesadores, y cada procesador puede ejecutar diferentes instrucciones en cualquier momento. Esta clase es en general la más usada.

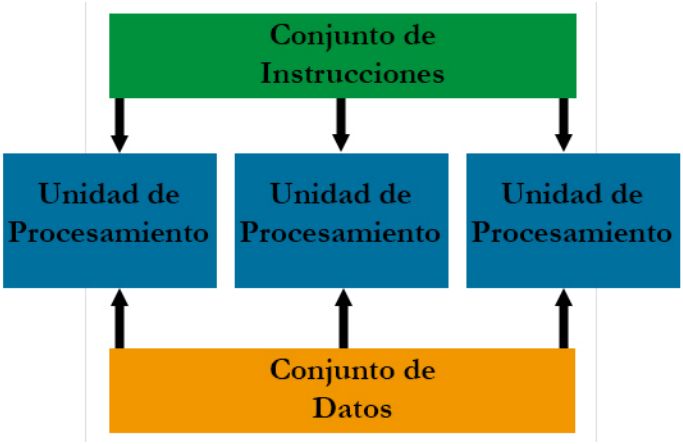


Figura 4 - Sistema MIMD

Los sistemas MIMD son subdivididos entre multiprocesadores, en los cuales todos los procesos tienen acceso directo a toda la memoria, y las multi-computadoras, también llamadas sistemas distribuidos, cada procesador tiene su propio módulo de memoria local y módulos de memoria remota accediendo mediante el uso de mecanismos de paso de mensajes.

Al día de hoy, los procesadores son paralelos en la forma en que ellos ejecutan las instrucciones, pero siguen siendo considerados SISD.

Los sistemas distribuidos están compuestos de colecciones de computadoras interconectadas, cada una teniendo su propio procesador, memoria y adaptador de red. Comparado con multiprocesadores, ellos tienen ciertas ventajas: fáciles de crear y extender, mejor relación precio/rendimiento, mayor escalabilidad y mayor flexibilidad [27]. Un sistema distribuido común es un clúster de computadoras interconectadas por una red de comunicaciones.

Los sistemas pertenecientes al modelo MPP (*Massively Parallel Processor*) son compuestos de miles de procesadores. Un MPP puede ser compuesto de máquinas que pertenecen a múltiples organizaciones y dominios, conduciendo los sistemas de malla, conocidas como meta-computadoras, las cuales son construidas alrededor de la infraestructura provista por internet.

3.2.1 Programación de memoria compartida y memoria distribuida

Comparado con la programación secuencial; la cual está basada en un simple proceso que tiene un único flujo de control; los programas de computadoras paralelas se les llama programación concurrente. Un programa concurrente tiene dos o más procesos que trabajan juntos para ejecutar una tarea comunicándose y sincronizándose entre ellos [28], es decir, el programador tiene que lidiar con la exclusión múltiple, condiciones de sincronización o los bloqueos mutuos (también conocidos como *deadlocks*) [4].

La programación de memoria compartida está basada en el hecho de que toda la memoria puede ser accedida por todos los procesadores mediante operaciones de lectura y escritura.

La programación distribuida está basada en mecanismos de paso de mensajes. Para empezar, existen diferentes formas de intercambiar mensajes entre los procesos, además el programa tiene que resolver problemas relacionados a la heterogeneidad (las maquinas que envían y reciben mensajes suelen tener diferentes arquitecturas o sistemas operativos), balance de carga (para mantener a todos los procesos ocupados como sea posible), seguridad, etc.

La forma más fácil de trabajar con paralelismo en memoria compartida es el uso de un compilador que automáticamente convierta un programa secuencial a uno paralelo. El segundo enfoque es usar recursos del sistema operativo como son los procesos, hilos, semáforos o inclusive archivos. Un tercer grupo es compuesto de librerías paralelas que pueden ser usadas por lenguajes secuenciales como OpenMP (www.openmp.org). Por último podemos usar lenguajes paralelos [4].

En cuanto a la programación de memoria distribuida, podemos considerar también el uso de recursos de sistema operativo (*Sockets)*, librerías paralelas (PVM, MPI) y lenguajes paralelos.

Existen un sinfín de herramientas con las que se cuenta hoy en día para poder trabajar con la memoria distribuida, las más comunes son [4]:

- **Librerías de paso de mensajes.** Estas librerías son especializadas en el desarrollo de aplicaciones en clústeres de computadoras. Se puede incluir *sockets* que ofrece el sistema operativo, sin embargo, la API de socket es una interfaz de bajo nivel, mientras que las

librerías como pueden ser PVM y MPI ofrecen un conjunto de primitivas para los procesos de comunicación y sincronización.

- **Sistemas basados en objetos.** Ellos son una evolución de los sistemas de llamadas a procedimientos remotos (RPC), y están basados en la idea de tener objetos remotos que puedan ser accedidos por clientes que invoquen métodos que se definen en las interfaces. Para controlar la heterogeneidad, los sistemas basados en objetos, cuentan con una estructura de tres capas, las cuales son: clientes y objetos (nivel superior), sistema operativo y hardware (nivel inferior) y el nivel intermedio llamado *middleware,* el cual oculta los detalles del nivel más bajo a las aplicaciones.
- **Sistemas de computación en malla.** Meta-computadoras compuestas de miles de máquinas pertenecientes a organizaciones que residen en diferentes partes del mundo. El poder computacional que puede ser obtenido por una malla de sistemas computacionales permite atacar problemas que no puedan ser resueltos por los sistemas de clústeres.
- **Computación basada en la web.** Mejor conocidos como servicios Web, proveen un método para poder comunicar aplicaciones entre ellas utilizando el internet. Comparadas con los sistemas en malla, los servicios web utilizan los protocolos web y los estándares XML.

3.2.2 Medidas de desempeño

Para poder determinar la mejora que se obtiene al paralelizar los algoritmos y determinar que tan bien se paraleliza, es necesario establecer una serie de medidas que nos permitan obtener esto. Las tres principales medidas son la aceleración, eficiencia y fracción serial.

Aceleración

La aceleración (*Speedup*) es la razón entre el tiempo de ejecución secuencial y el tiempo de ejecución en paralelo [29].

$$S_m = \frac{T_1}{T_m}$$

Donde:

S_m es la aceleración en *m* unidades de procesamiento

T_1 es el tiempo de ejecución en una unidad de procesamiento

T_m es el tiempo de ejecución en *m* unidades de procesamiento

Para algoritmos no deterministas estas métricas no se puede usar directamente. Para este tipo de algoritmos se debe comparar el promedio del tiempo de ejecución serial entre el promedio del tiempo de ejecución en paralelo [30].

Ley de Amdahl

Un modelo más general para medir la aceleración fue propuesto por Gene Amdahl, el cual es mejor conocido como la *Ley de Amdahl* [31].

El modelo de Amdahl establece [32] que

$$S_p = \frac{1}{\xi + \frac{1-\xi}{P}}$$

Donde:

S_p es la aceleración

ξ es el porcentaje serial del código

P es el porcentaje en paralelo

En su forma más simple se asume que algún porcentaje del programa o código no puede ser paralelizado. Despreciando cualquier tipo de retrasos en la comunicación, contención de memoria, latencia, etc.

Eficiencia

La eficiencia de un programa paralelo es una medición de la utilización de procesadores en comparación con la cantidad de esfuerzo desperdiciado en comunicación y sincronización. Definimos la eficiencia como

$$e_m = \frac{S_m}{m}$$

Donde:

e_m es la eficiencia con m unidades de procesamiento

m es el número de unidades de procesamiento

S_m es la aceleración en *m* unidades de procesamiento

Fracción serial

La fracción serial es una métrica propuesta por Karp y Flatt [33] que nos ayuda a estimar la fracción del tiempo de ejecución serial en un algoritmo paralelo.

La fracción serial se define como:

$$f_m = \frac{\frac{1}{S_m} - \frac{1}{m}}{1 - \frac{1}{m}}$$

Donde:

m es el número de unidades de procesamiento

S_m es la aceleración en *m* unidades de procesamiento

f_m es la fracción serial en *m* unidades de procesamiento

Si la aceleración de un algoritmo es pequeña, aun podemos decir que es eficiente si f_m permanece constante para diferentes valores de m. Si el valor de f_m aumenta con el número de procesadores, entonces se considera un indicador de escalabilidad pobre [34].

3.3 Paralelización de Metaheurísticas

En la práctica los problemas de optimización son, la mayoría de las veces *NP-hard*, complejos y consumen una gran cantidad de tiempo del CPU. Para solucionar estos problemas existen dos métodos tradicionalmente usados: los métodos exactos y las Metaheurísticas. Los métodos exactos permiten encontrar soluciones exactas, pero el problema es que son imprácticos debido al excesivo consumo de tiempo que requieren para dar solución a los problemas. Por otro lado las metaheurísticas permiten encontrar soluciones casi óptimas en un tiempo razonable [35].

Aunque el uso de Metaheurísticas permite significativamente reducir la complejidad temporal del proceso de búsqueda, en algunas aplicaciones de la vida real consumen una gran cantidad de tiempo. Es por ello que el paralelismo es necesario no solo para reducir el tiempo que tarda en obtener una solución, sino que también permite mejorar la calidad de las soluciones.

Las metaheurísticas basadas en poblaciones siguen dos enfoques: paralelización de la computación y paralelización de poblaciones. En el primer modelo, las principales operaciones aplicadas a cada individuo son paralelizadas. En el segundo modelo, la población es dividida en diferentes subpoblaciones que pueden evolucionar separadamente y unirse después [35].

3.3.1 Algoritmos evolutivos paralelos

Los algoritmos evolutivos (AEs) son técnicas de búsqueda estocásticas que han sido aplicadas en muchos problemas reales y aplicaciones complejas. Estos AEs pertenecen al campo del *Cómputo Evolutivo,* los cuales son métodos computacionales inspirados por procesos y mecanismos de la evolución biológica.

Los AEs comparten propiedades de adaptación mediante un proceso iterativo que guarda y mejora el mejor resultado mediante operadores estocásticos. Las soluciones candidatas representan miembros de una población que se esfuerza por sobrevivir en un ambiente definido por una función objetivo de un problema específico. El proceso evolutivo redefine la aptitud de las soluciones candidatas de la población, típicamente utilizando mecanismos de evolución como recombinación genética y mutación [36].

```
AE pseudocódigo
Generar(P(0));
t := 0;
Mientras no Criterio_Termino(P(t)) hacer
    Evaluar(P(t));
    P'(t) := Seleccion(P(t));
    P'(t) := Aplicar_Operaciones_Reproduccion(P'(t));
    P(t+1) := Reemplazar(P(t),P'(t));
    t := t + 1;
FinMientras
```

Algoritmo 1 - Algoritmo Evolutivo

3.3.2 Modelos paralelos de AEs

La ejecución del ciclo de reproducción de un simple AE de una gran cantidad de individuos y de poblaciones requiere grandes recursos computacionales. En general, la evaluación de la función de aptitud de un EA para cada individuo es la función de mayor costo.

El paralelismo surge naturalmente al trabajar con poblaciones. Ya que cada individuo pertenece a una unidad independiente y es posible poder mejorar el rendimiento cuando se ejecuta en paralelo.

Se han desarrollado diferentes métodos de paralelilzación de los algoritmos AEs. El primero de ellos es el método Maestro-Esclavo, en el cual un procesador central controla las operaciones de selección mientras que los procesadores esclavos controlan las operaciones de recombinación, mutación y evaluación de la función aptitud [35].

Actualmente la mayoría de los métodos paralelos usados en AEs utilizan una distribución de los individuos en un conjunto de procesadores. Entre los más conocidos son los tipos estructurados con son los Algoritmos Evolutivos Distribuidos (AEd) y los celulares (AEc) [37].

En los AEds, la población es partida en un conjunto de islas las cuales son ejecutadas aisladamente. Entre las islas existe un intercambio de individuos con el objetivo de introducir diversidad entre las poblaciones y de esta forma evitar caer en óptimos locales.

Para el caso de los AE celulares, se introduce el concepto de vecindario, el cual es un conjunto de individuos que solo interactuar con sus vecinos más cercanos. El traslape entre los vecindarios provee un tipo de exploración, mientras que la explotación se realiza entre cada vecindario.

3.3.3 Algoritmos Genéticos Paralelos

Los algoritmos genéticos (AGs) [38] son una subfamilia de los algoritmos evolutivos (AEs). Estos algoritmos son métodos de búsqueda estocásticos diseñados para explorar en espacios de problemas complejos, con el objetivo de encontrar soluciones óptimas usando mínima información. A diferencia de otras técnicas de optimización, los AGs son caracterizados por usar una población de múltiples estructuras (individuos) para mejorar la búsqueda a través de diferentes áreas del problema en un mismo tiempo. Los individuos son codificaciones de posibles soluciones a los cuales se les aplica operadores estocásticos para encontrar una solución, si no global, una solución satisfactoria.

Un esbozo de un AG clásico se describe a continuación.

AG Clásico pseudocódigo

```
Generar(P(0));
Evaluar(P(0));
t := 0;
Mientras no Criterio_Termino(P(t)) hacer
    P'(t) := Seleccion(P(t));
    P''(t) := Recombinacion(P'(t));
    P'''(t) := Mutacion(P''(t));
    Evaluar(P'(t));
    P(t+1) := Reemplazar(P(t),P'''(t));
    t := t + 1;
FinMientras
```

Algoritmo 2 - Algoritmo genético clásico

Un AG genera una nueva población *P(t)* de individuos a partir de una población *P(t-1)(t = 1,2,3,..).* La población inicial *P(0)* es generada aleatoriamente. Una función de aptitud asocia un valor a cada individuo.

El algoritmo clásico aplica operadores estocásticos como la selección, cruza y mutación en una población con el objetivo de obtener nuevos individuos. Se aplican operadores de variación para crear una población temporal *P'(t)* los cuales se evalúan. Después una nueva población *P(t+1)* es obtenida usando *P'(t)* . El criterio de paro es usualmente es un número de iteraciones del algoritmo y/o encontrar un individuo aproximado siempre y cuando se conozca este.

3.3.4 Elementos de un algoritmo genético.

La representación del individuo

Denominamos cromosoma a una estructura de datos que contiene una cadena de parámetros de diseño o genes. Esta estructura de datos puede ser representada, como muchos autores la expresan, como una cadena de bits, sin embargo la tendencia actual es que se exprese en números reales. Estas cadenas representan el genotipo que codifica las variables de decisión de un problema.
La eficacia de un AG para resolver un problema depende fundamentalmente de la representación que se haga de los genes, cromosomas e individuos de una población [14].

Inicialización

La población inicial es usualmente generada aleatoriamente, aunque también es fácil usar el conocimiento del problema o algún otro tipo de información.

Evaluación

Una vez que se ha inicializado la población o cuando una nueva solución de una nueva generación es creada, es necesario calcular el valor de aptitud de la soluciones candidatas.

Selección

Es importante el proceso de selección, ya que permite poder seleccionar de la mejor manera a los individuos candidatos a reproducirse.

Hoy en día, el proceso de selección tiende a ser de manera probabilística, permitiendo que los individuos menos aptos tengan cierta oportunidad de sobrevivir y transmitir las cualidades buenas que posean.

Los métodos más comunes de selección son: Selección por ruleta y selección por torneo.

Método de la ruleta

Consiste en asignar a cada individuo de la población un porcentaje a su ajuste en la ruleta, de tal forma que la suma de todos los porcentajes sea el 100%. Los individuos más aptos reciben un porcentaje alto en la ruleta [39].

Método por torneo

Este método consiste en realizar la selección con base en comparaciones directas entre individuos. Existen dos versiones de este método: Determinística y probabilística.

La versión determinística selecciona al azar un número de individuos. De entre los individuos se selecciona el más apto.

La versión probabilística se genera un número aleatorio de intervalo 0-1, si es mayor que un parámetros establecido, se escoge el individuo más apto, en caso contrario el menos apto.

Cruza

La función de cruza intercambia segmentos de cadenas de longitud fija entre dos individuos permitiendo obtener un nuevo individuo con propiedades parecidas a las de los padres.

Optimización numérica

- Cruza de números reales (*Flat crossover*)

  Dados dos padres donde X y Y representan un número flotante:
  - Padre 1: X
  - Padre 2: Y
  - Hijo: rnd(X,Y)

Optimización combinatoria

- Cruza ordenada

Dados dos cromosomas padres, se generan dos puntos aleatorios b1 y b2, dichos puntos dividen al cromosoma en tres partes: izquierda, central y derecha. El hijo uno hereda la parte izquierda y derecha del padre uno, y la selección del centro es determinada por los genes de la parte central del padre uno en el orden en el cual aparecen en el padre dos. Para generar el hijo dos, se aplica el mismo procedimiento.

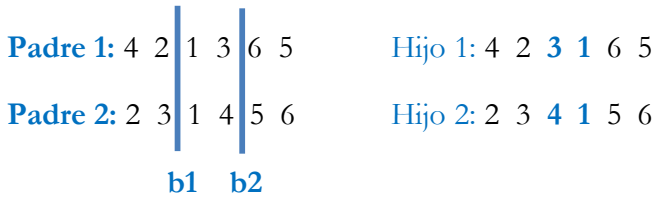


Figura 5 - Ejemplo de cruza ordenada

- Cruza basada en posición
    1. Elegir aleatoriamente un conjunto de posiciones del padre P1.
    2. Generar un hijo H1 eliminando de P1 todos los valores excepto los elegidos en el paso uno.
    3. Borrar los valores seleccionados en el paso uno del padre P2.
    4. Colocar en H1 los valores faltantes de izquierda a derecha de acuerdo a la secuencia de P2.

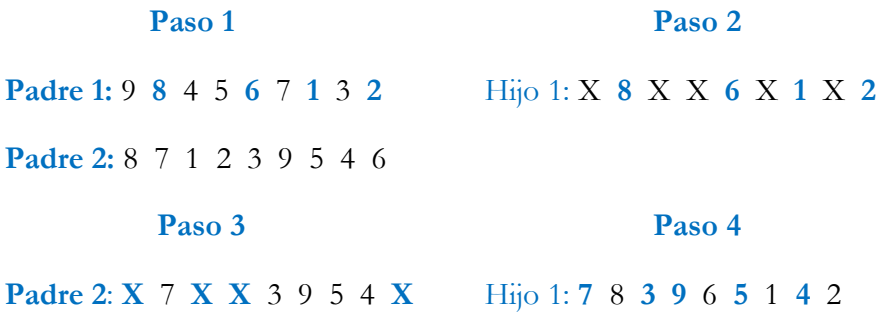


Figura 6 - Ejemplo de cruza basada en posición

Mutación

El operador de mutación tiene como objetivos explorar áreas posiblemente no abordadas del espacio de búsqueda (cercanas a una buena solución) y sacar al GA de un máximo local si se produjo convergencia prematura.

Optimización numérica

- Mutación gaussiana

  A los individuos generados a partir de la cruza les sumamos un valor aleatorio con distribución normal.

  Dados dos padres donde X y Y representan un número flotante:

  - Padre 1: X

  - Padre 2: Y

  - Hijo: $rnd(X,Y)+ \sigma N(0,1)$

Optimización combinatoria

- Mutación por inserción

  Se selecciona aleatoriamente una posición b1 y se inserta en una posición b2

b1: 8 **b2:** 2

Individuo: 9 8 4 5 6 7 1 **3** 2

Individuo: 9 **3** 8 4 5 6 7 1 2

Figura 7 - Ejemplo de mutación por inserción

- Mutación por intercambio recíproco

  Se seleccionan aleatoriamente dos posiciones b1 y b2 y se intercambian sus valores.

b1: 7 **b2:** 3

Individuo: 9 8 **4** 5 6 7 **1** 3 2

Individuo: 9 8 **1** 5 6 7 **4** 3 2

Figura 8 - Ejemplo de mutación por intercambio recíproco

Reemplazo

Los nuevos individuos creados a partir de la selección, cruza y mutación, reemplazan a los padres de acuerdo a un criterio que se establezca; el reemplazo debe preservar la mejor solución de toda la población.

Para problemas no triviales, la ejecución del ciclo de reproducción de un simple AG requiere bastantes recursos computacionales (por ejemplo: mucha memoria y un tiempo muy grande de búsqueda).

Los algoritmos genéticos paralelos (AGP) han tomado mucha popularidad y se han desarrollado un gran número de implementaciones y algoritmos debido a que los AG son naturalmente paralelizables.

3.3.5 Algoritmos genéticos estructurados

En las implementaciones paralelas de AG, tradicionalmente se utiliza una estructura en las poblaciones. Entre las estructuras de AG más conocidas se encuentran [40]:

- El algoritmo genético distribuido (AGd)
- El algoritmo genético celular (AGc)

Una población simple puede ser mejorada partiéndola en diferentes subpoblaciones (islas), en la que cada isla AG realiza intercambio (migración) de individuos (AGd) o en forma de vecindarios (AGc).

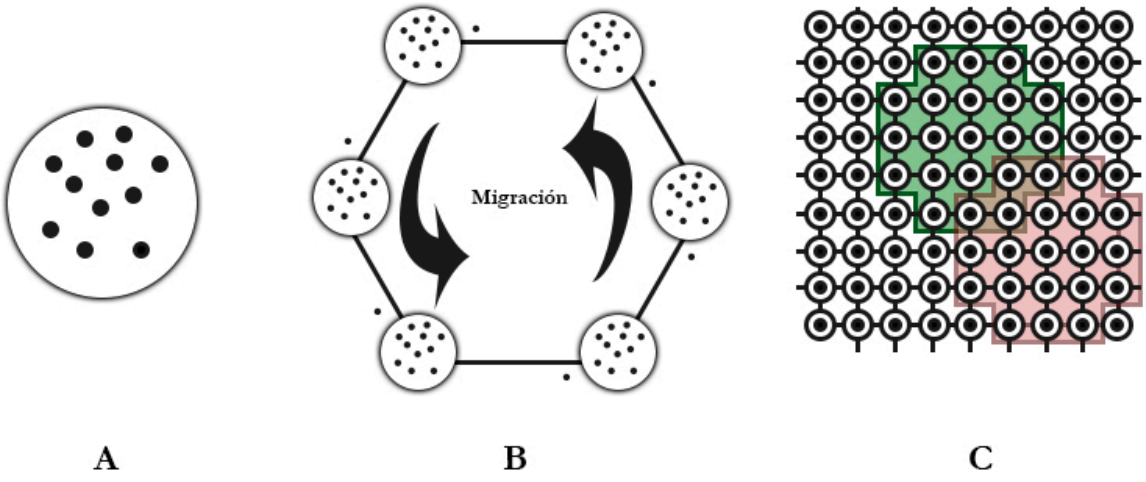


Figura 9 - (A) Una sola población, (B) AGd, (C) AGc

En los AGd, parámetros adicionales controlan cuando ocurre la migración y como los migrantes son seleccionados, incorporados, origen y destino, etc [41]. Este tiene mucha popularidad por la fácil implementación que tiene en un sistema con memoria distribuida (MIMD).

En los AGc, el hecho de que los vecindarios se traslapen, ayuda en la exploración del espacio de búsqueda. Un AGc puede ser implementado en una computadora con memoria distribuida (MIMD) aunque obtenemos una mejor eficiencia en una computadora con memoria compartida (SIMD)

Estos dos tipos de AG ayudan a mejorar la búsqueda y mejorar el tiempo de ejecución en muchos casos.

3.3.6 Modelos de paralelización

Modelo de ejecuciones independientes

Este modelo consiste meramente en la ejecución en paralelo un mismo algoritmo secuencial, sin interacción entre las ejecuciones independientes.

Este modelo puede ser usado para ejecutar el mismo problema pero con diferentes condiciones iniciales, permitiendo obtener estadísticas de estas.

Modelo maestro-esclavo

Este modelo consiste en la distribución de la función de evaluación entre un conjunto de procesadores esclavos mientras la ejecución principal del GA es ejecutada en un procesador maestro.

El procesador maestro controla la paralelización de las tareas de la función de evaluación y posiblemente la asignación de la aptitud y/o transformación entre los procesos esclavos.

Figura 10- Modelo Maestro-Esclavo

Modelo distribuido

En este modelo, la población es estructurada en pequeñas poblaciones también llamadas *islas,* aisladas unas de otras. Los AGP basados en este modelo son algunas veces llamados Algoritmos genéticos multipoblacionales.

La principal característica de este tipo de algoritmo es que individuos dentro de una particular subpoblación (isla) puede ocasionalmente migrar a otra subpoblación.

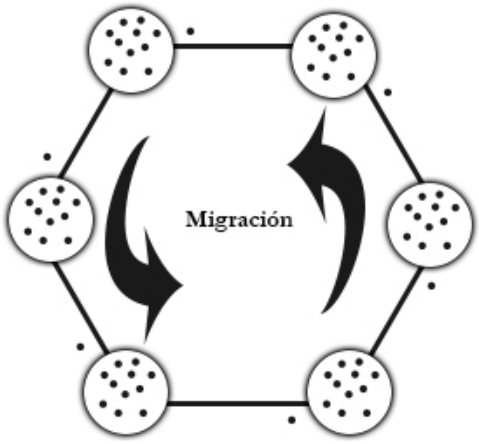


Figura 11- Modelo distribuido

En este modelo, las operaciones genéticas (selección, mutación y recombinación) se realizan dentro de cada isla, lo que significa que cada isla puede buscar en diferentes regiones de todo el espacio de búsqueda, así como tener diferentes parámetros.

El modelo distribuido requiere de una política de migración. Los principales parámetros de la política de migración son los siguientes:

- **Margen de migración**: Es el número de ejecuciones dentro de una subpoblación para poder realizar el intercambio.
- **Razón de migración**: Este parámetro determina el número de individuos que participaran en la migración.
- **Selección/Reemplazo de migrantes**: Decide como seleccionar los emigrantes y cuales habrán de ser reemplazados.
- **Topología**: Define como será la conexión de cada isla.

Modelo celular

La principal característica de este modelo es la estructura de la población en vecindarios, y los individuos solo pueden interactuar con sus vecinos.

Particularmente, los individuos están conectados en una malla toroidal y solo se permite recombinarse con individuos cercanos [42].

En este modelo la población usualmente está estructurada en una malla bidimensional de individuos como se muestra en la figura siguiente.

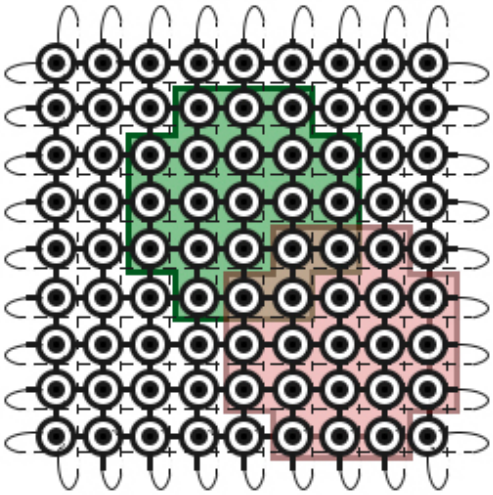


Figura 12- Modelo celular

En esta malla los individuos están conectados a los individuos localizados en los bordes opuestos en la misma fila/columna, según sea el caso. El efecto de esta malla toroidal, es que los individuos tienen exactamente el mismo número de vecinos.

El vecindario de un individuo se define en términos de la distancia Manhattan entre el individuo y los otros en la población. Cada individuo de la malla tiene un vecindario que se traslapa a otros vecindarios de sus individuos más cercanos lo que permite proveer un mecanismo de migración. Todos los vecindarios tienen el mismo tamaño y la misma forma.

En la figura siguiente podemos ver las principales formas de vecindarios más usados en este modelo.

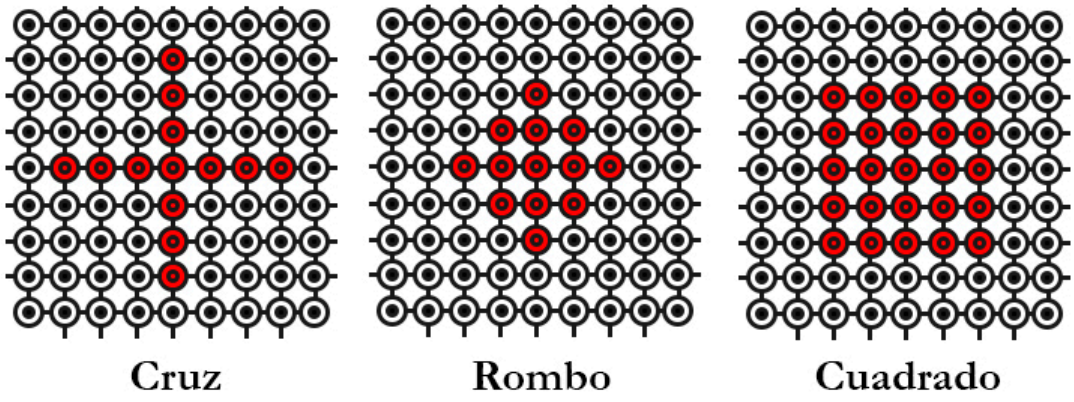


Figura 13 - Formas de vecindarios

En los AGc, los individuos solamente pueden interactuar con sus vecinos, esto quiere decir que el algoritmo genético es ejecutado dentro del vecindario de cada individuo y generalmente consiste en la selección de dos padres dentro de este vecindario de acuerdo a cierto criterio, en seguida aplicar los operadores de cruza y mutación y reemplazar el nuevo individuo generado siempre y cuando sea mejor que el individuo considerado.

En la figura se puede observar como el ciclo de reproducción es aplicado en el vecindario de un individuo en un AGc.

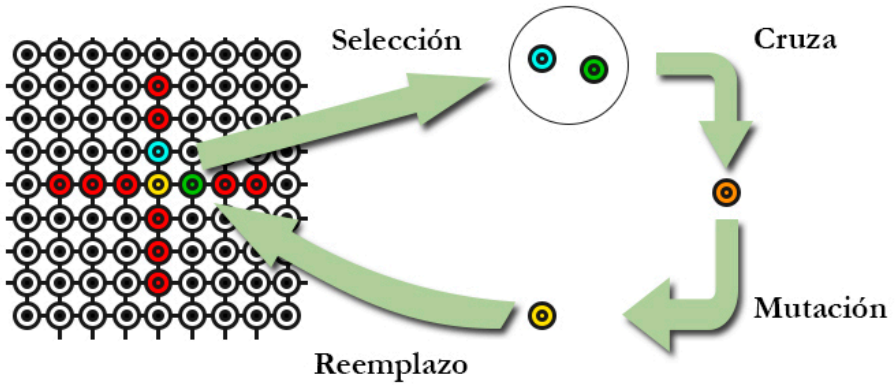


Figura 14- Ciclo de un AGc

En este modelo debemos tomar una política de actualización en la población, la cual puede ser síncrona o asíncrona.

AGc Síncronos y Asíncronos

Existen dos tipos de AGc acorde a como el ciclo de AG es aplicado a los individuos.

Si el ciclo es aplicado a todos los individuos simultáneamente, se dice que el AGc es síncrono, es decir, los individuos de la población de la siguiente generación son creados al mismo tiempo en forma concurrente.

Por otro lado, si nosotros actualizamos los individuos de la población secuencialmente con alguna cierta política, se dice que el AGc es asíncrono.

Capítulo 4
MÉTODO PROPUESTO E IMPLEMENTACIÓN

En el capítulo anterior se explicó a detalle los diferentes modelos de metaheurísticas multipoblacionales que se han propuesto en la literatura, así como las tecnologías de paralelismo más actuales.

En este capítulo se explica la biblioteca desarrollada llamada MPHStudio, la cual permite realizar experimentos con algoritmos genéticos multipoblacionales en paralelo. Se explicará las implementaciones multipoblacionales y las tecnologías utilizadas en estas.

4.1 Algoritmos genéticos

Para el desarrollo de este trabajo, se optó por implementar el algoritmo genérico, en primera parte por ser uno de los algoritmo más estudiados, documentados y con muy buenos resultados en la búsqueda de soluciones a problemas de optimización; y en segunda parte debido a que este es uno de los algoritmos que más operaciones tiene así como la cantidad de parámetros que requiere es mayor en comparación con otros algoritmos. Aunque pudiera parecer desventaja lo que se menciona, el hecho de que este algoritmo tenga varias operaciones y varios parámetros, nos sirve para hacer la biblioteca más robusta.

4.1.2 Paralelización del algoritmo genético distribuido

La primera implementación que se explicará, es el algoritmo genético distribuido, en este algoritmo, como se mencionó en el capítulo anterior, se estructura en pequeñas subpoblaciones las cuales son llamadas islas.

En esta implementación; con base en las características de la tecnología, las características de este algoritmo, la documentación proporcionada y la experiencia previa; se decidió utilizar MPI, el cual trabaja con un modelo de memoria distribuida.

En la implementación propuesta, cada una de las islas evoluciona dentro de un proceso creado por la biblioteca MPI, y solo se comunican entre ellas cuando el proceso de migración es ejecutado. Este proceso trabaja con base en los parámetros de migración que se establecen al inicio de la ejecución del algoritmo.

La cantidad máxima de islas que pueden ser ejecutadas a la vez, va a depender completamente de las capacidades del sistema de cómputo y de las políticas del sistema que se establezcan en cuestión de ejecución de procesos de MPI.

El modelo de paralelización está directamente relacionado con la topología de conexión establecida en la política de migración del modelo distribuido, es decir, la comunicación y la sincronización entre los procesos que controlan las islas se realizará dinámicamente según la topología de conexión. En la figura 15 y 16 se muestra un ejemplo de una topología de anillo y una topología arbitraria.

Además de los procesos que se encargan de ejecutar el proceso de evolución en cada una de las islas, existe un proceso adicional que es el encargado de administrar el método de paro de la ejecución del algoritmo, generación de la población inicial, así como el procesamiento de los resultados finales.

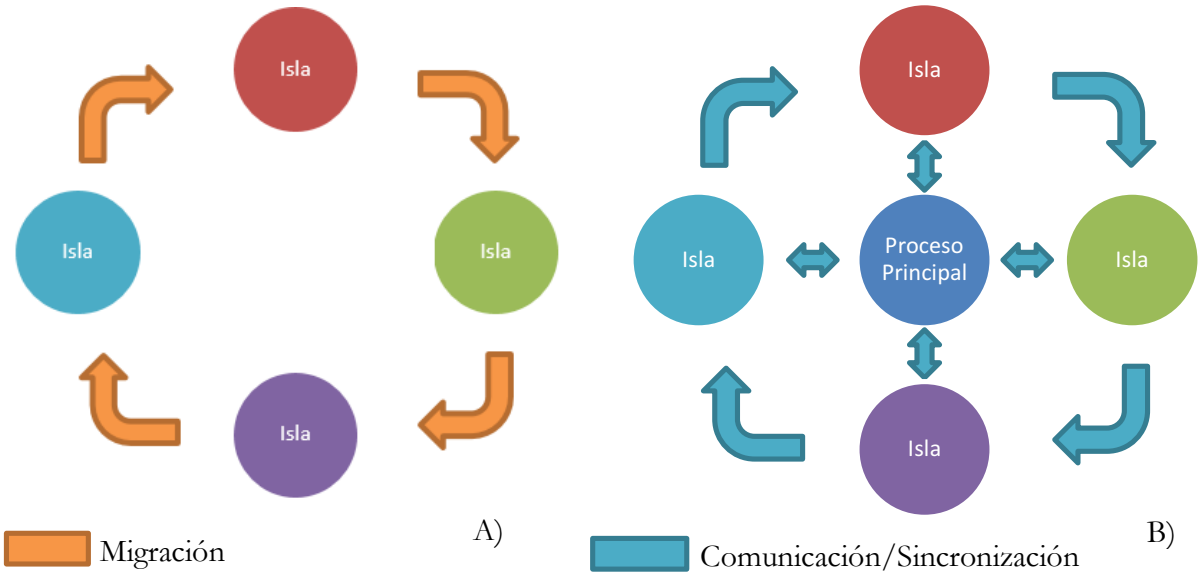


Figura 15 – Topología de anillo. A) Modelo de islas. B) Modelo de paralelización

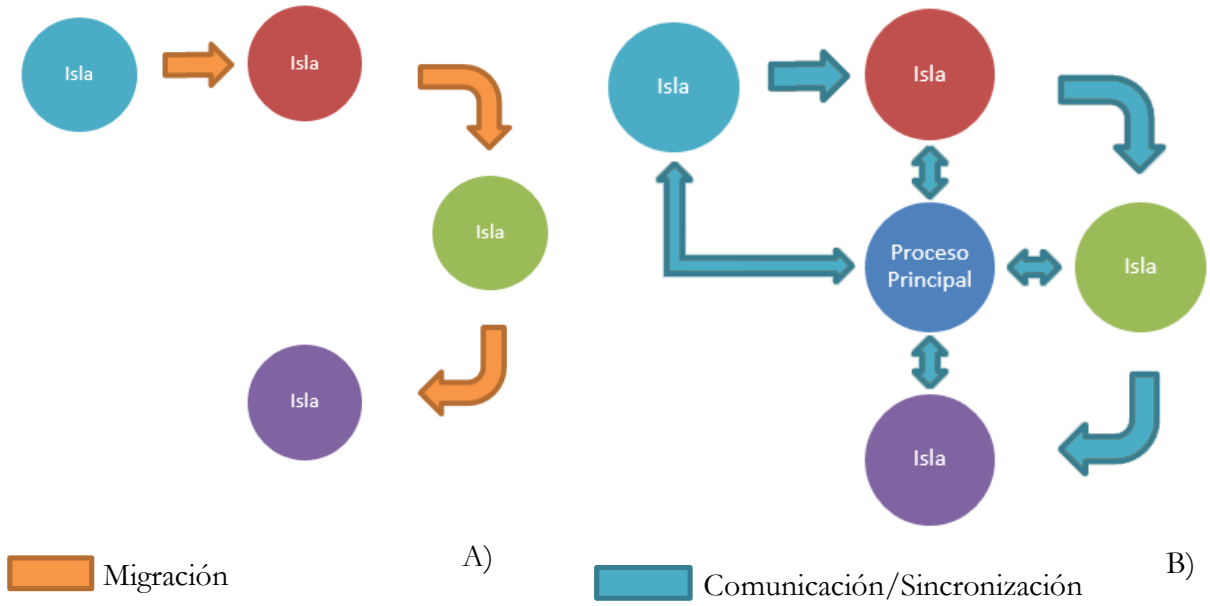


Figura 16 - Topología arbitraria. A) Modelo de islas. B) Modelo de paralelización

El motivo del uso de un proceso adicional a la cantidad de islas requeridas, es poder garantizar el paro de todos los procesos casi al mismo tiempo, así como la correcta sincronización de los procesos. De no hacerlo así, puede suceder que alguno de los procesos se quedé esperando la respuesta de otro proceso que ya haya sido detenido y provoque un bloqueo mutuo. Un ejemplo de sincronización entre los procesos se puede observar en la siguiente imagen.

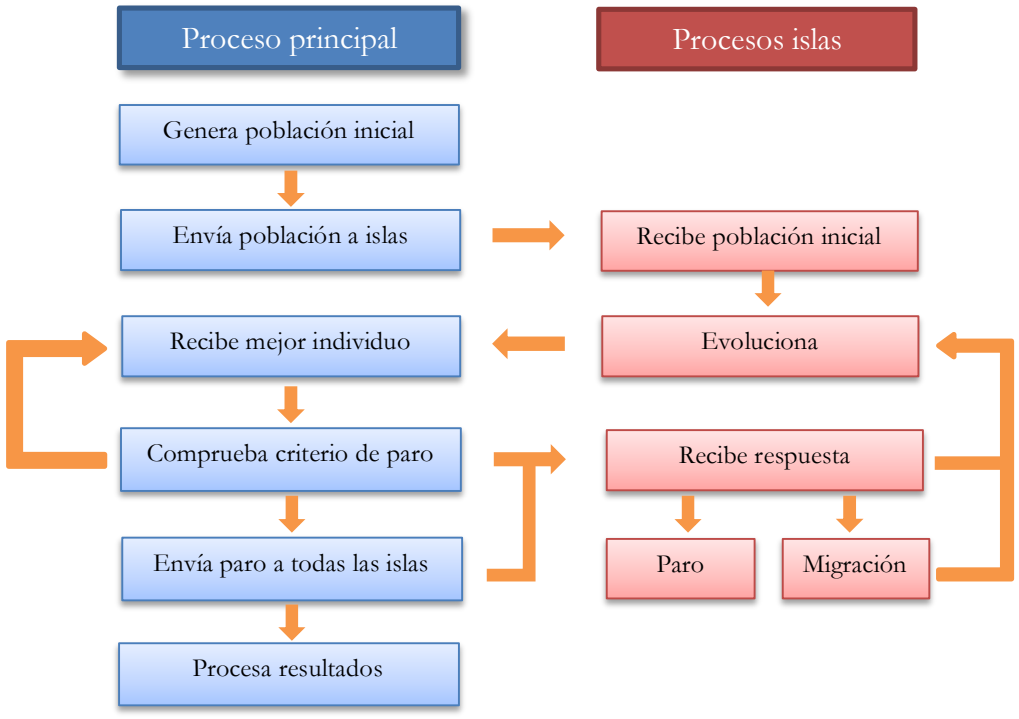


Figura 17 - Diagrama de sincronización entre procesos del modelo distribuido

Según se especifique en los parámetros de entrada, cada uno de los procesos que controlan las islas va registrando el resultado de cada una de las operaciones del AG y los individuos que son migrados a las otras islas. Esto nos permite hacer un seguimiento del comportamiento del experimento y si se requiere, modificar los parámetros que permitan mejorar el proceso de optimización.

Además de lo ya mencionado, se ha agregado un método de escape dentro de la ejecución del AG de cada isla, que nos permite generar nuevos individuos aleatoriamente e introducirlos dentro de la población para poder escapar de los óptimos locales.

El proceso principal también es el encargado; según se requiera; de guardar los resultados en un archivo, guardar la población inicial para después poder ser cargada de nuevo, graficar los resultados de la ejecución del algoritmo y crear el archivo de seguimiento de ejecución.

4.1.3 Paralelización del algoritmo genético celular

La siguiente implementación de algoritmo genético multipoblacional, es el algoritmo genético celular, el cual como se mencionó en el capítulo anterior, su principal característica es la estructura de la población en vecindarios y la interacción entre individuos solo se realiza entre vecinos. Con base en esta característica se decidió utilizar como tecnología de paralización hilos, ya que la ventaja de la utilización de hilos es que debido a que los hilos comparten memoria y recursos, hacen más rápida la comunicación entre ellos.

En python la gestión de hilos es controlada por el *GIL* (Global Interpreter Lock) de forma que solo un hilo pueda ejecutarse a la vez. La ventaja de esto es la posibilidad de escribir más fácil extensiones de C para python [43].

Cada cierto número de instrucciones de bytecode, la máquina virtual detiene la ejecución del hilo y elige algún otro que se encuentra en espera.

Debido a esta limitante, se recurrió al multiprocesamiento en python.

En python el multiprocesamiento ofrece concurrencia local y remota, utilizando subprocesos en lugar de hilos. Este módulo permite al programador aprovechar plenamente múltiples procesadores en un equipo determinado [44].

En el método propuesto, un subproceso controla por lo menos a un vecindario. Cuando hay más individuos que subprocesos, los subprocesos controlan bloques de vecindarios. En la siguiente imagen se muestra la forma en que los procesos son repartidos para controlar los diferentes individuos que son centros en sus vecindarios correspondientes.

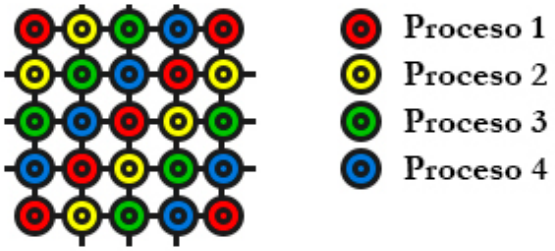


Figura 18 - Modelo celular y sus procesos

El ejemplo de la imagen es una estructura celular de 25 individuos y 4 subprocesos que la controlan. En ese caso el número de procesos es mucho menor al número de individuos, por lo tanto algunos procesos tienen que controlar más de un vecindario.

Es de suma importancia mencionar que un subproceso no puede controlar a más de un vecindario al mismo tiempo, eso quiere decir, siguiendo este ejemplo, que solo estarán evolucionando 4 vecindarios concurrentemente.

El proceso de ejecución principal es el encargado de reemplazar los individuos dentro de la población de la malla, así como el control del proceso de paro del algoritmo dependiendo de los parámetros con el cual se ejecutó.

En la siguiente imagen se puede observar un diagrama de sincronización entre el proceso principal y los procesos que controlan a los vecindarios.

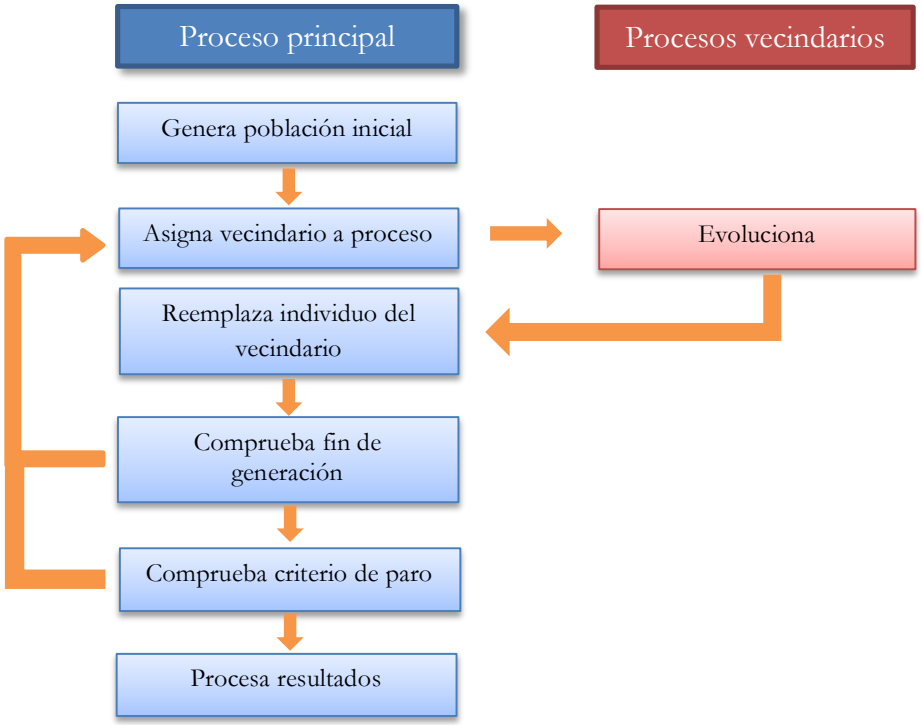
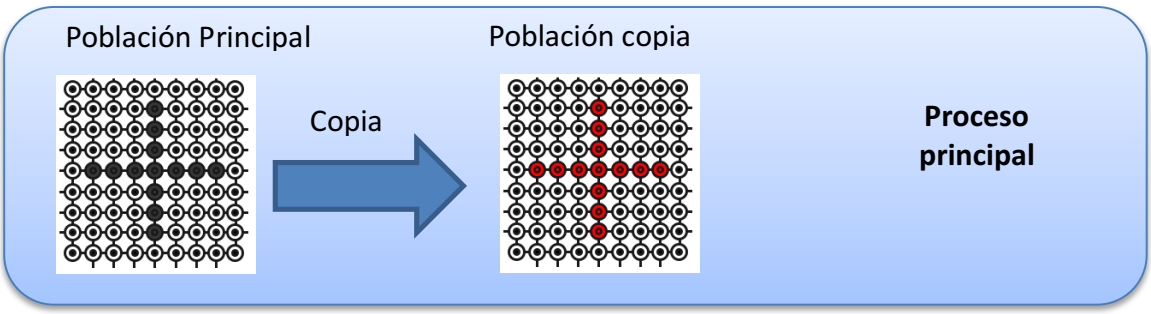


Figura 19 - Diagrama de sincronización entre procesos del modelo celular

El tipo de AGc lo determina el usuario por medio de los parámetros de entrada. Cuando se elige un AGc síncrono, el proceso principal crea una copia de la malla de individuos principal al inicio de cada generación con la finalidad de que durante cierta evolución, todos los vecindarios vean los mismos valores sin importar si todos los vecindarios evolucionan concurrentemente o van evolucionando en grupos.



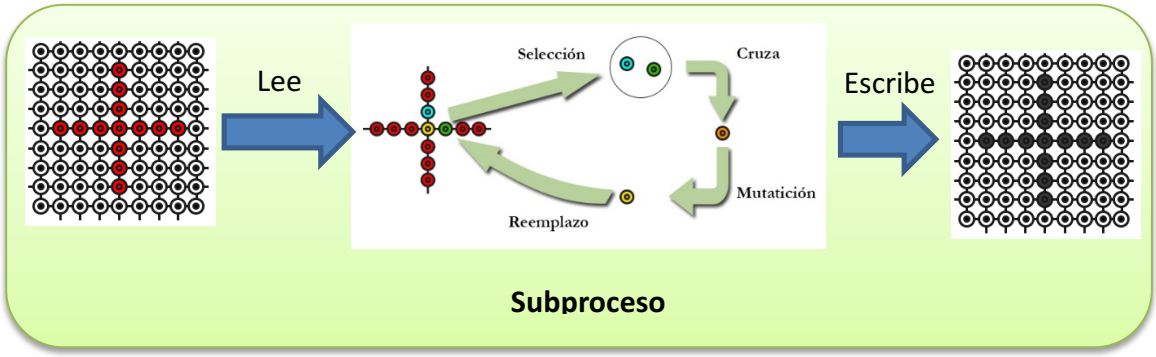


Figura 20 - Modelo del algoritmo genético celular síncrono

El proceso principal se encarga de registrar los eventos que van sucediendo dentro del proceso de cada una de las evoluciones del algoritmo genético celular.

Cabe destacar que así como en la implementación del modelo de islas, se implementó una función que permite escapar del óptimo local, ya que existen problemas que por su complejidad es difícil poder realizar una buena exploración del espacio de búsqueda. Esta función comprueba que si en un lapso de generaciones no ha habido mejora en la aptitud del individuo central del vecindario se generan nuevos vecinos aleatoriamente.

4.2 MPHStudio

Se desarrolló una biblioteca que tiene implementado el algoritmo genético distribuido en paralelo y el algoritmo genético celular en paralelo. La biblioteca tiene el nombre de MPHStudio (*Multi-Population Heuristic Studio*). Esta biblioteca cuenta con diferentes módulos que permiten ejecutar experimentos con diferentes parámetros de configuración y ejecución; además la biblioteca cuenta con herramientas que nos permiten obtener estadísticas de las ejecuciones que se realizaron, comparación entre diferentes configuraciones, seguimiento de ejecución, es decir, podemos saber cómo se está comportando el algoritmo con los parámetros que se establecieron previamente.

Además de todas estas características, es posible utilizar la biblioteca en desarrollos propios simplemente haciendo uso de la API desarrollada por nosotros.

La biblioteca MPHStudio fue desarrollada en lenguaje Python ya que este lenguaje cuenta con una serie de ventajas:

- Lenguaje de programación multiplataforma (Linux, Mac OS, Windows)
- Existe una amplia comunidad de desarrollo que han aportado una gran cantidad de bibliotecas, las cuales facilitan el desarrollo de muchas aplicaciones.
- La curva de aprendizaje es sencilla por el tipo de sintaxis que maneja, ya que asemeja un pseudocódigo.
- En los últimos años la comunidad científica ha adoptado este lenguaje de programación.
- Etc.

Como se mencionó con anterioridad, en esta biblioteca se implementó dos tipos de algoritmos genéticos multipoblacionales en paralelo; la implementación distribuida utiliza MPI como tecnología de implementación y la implementación celular utiliza multiprocesos. Más adelante se explicará la forma en que estos algoritmos fueron implementados así como las herramientas que contiene esta biblioteca.

4.2.1 Interfaz de MPHStudio

Como se mencionó con anterioridad la biblioteca MPHStudio fue implementada utilizando el lenguaje de programación Python. Para este lenguaje de programación existen una gran variedad de bibliotecas y herramientas que permiten diseñar y desarrollar interfaces gráficas, cada una con sus ventajas y desventajas.

Entre las más usadas son [44]:

- **Tkinter.** Es un módulo multiplataforma utilizando el toolkit Tk. Python incluye este toolkit por defecto. Se distribuye bajo licencia PSFL.
    - **Ventajas:** Popularidad, sencillez, documentación.
    - **Desventajas:** Herramientas, integración con el sistema operativo, lentitud.

- **WxPython.** Es un wrapper open source para el toolkit anteriormente conocido como wxWindows: wxWidgets. Ofrece un muy buen aspecto en todas las plataformas, utilizando

MFC en Windows y GTK en Linux. Se distribuyen bajo una licencia "wxWindows Licence", que consiste esencialmente en una LGPL.

- **Ventajas:** Popularidad, herramientas, multiplataforma.
- **Desventajas:** API poco apegada al estilo de desarrollo de python.

- **PyGTK.** Este módulo utiliza el toolkit GTK, biblioteca para desarrollar Gnome. Cuenta con grandes herramientas para construir la interfaz de forma gráfica, como Glade o Gazpacho. se distribuye bajo licencia LGPL.
  - **Ventajas:** Popularidad, Sencillez, herramientas.
  - **Desventajas:** Ligeramente más complicado de instalar y distribuir en Mac OS.
- **PyQT.** Este es un binding de Qt. Es una biblioteca multiplataforma que utiliza widgets nativos para las distintas plataformas. Cuenta con una herramienta muy completa para el diseño y creación de la interfaz llamada Qt Designer. PyQt utiliza un modelo de licencias similar al de Qt, con una licencia dual GPL/PyQt Commercial.
  - **Ventajas:** Sencillez, herramientas, multiplataforma, documentación.
  - Desventajas: Licenciada.

Ya mencionadas cada una de las bibliotecas más populares para el desarrollo de interfaces gráficas utilizando el lenguaje de programación python, se optó para el desarrollo de la biblioteca MPHStudio, la biblioteca gráfica PyQt. No solo por las ventajas antes mencionadas, sino además por la experiencia que se tiene en el manejo y desarrollo de aplicaciones creadas con Qt.

En esta biblioteca se ejecutan cada uno de los algoritmos que se han implementado, así como algunas herramientas que nos permiten realizar un análisis y seguimiento de las ejecuciones que se realicen en los experimentos.

4.2.2 Interfaz del algoritmo genético distribuido

La ejecución de este algoritmo requiere de ciertos parámetros de ejecución que nos permite ajustar el comportamiento del algoritmo; además de algunos otros parámetros que nos servirán de soporte para observación y seguimiento de los experimentos.

Los parámetros requeridos son:

- Islas
    - Número de islas.
    - Topología de conexión entre las islas.
    - Número de generaciones sin resultados mejores para el paro
    - Número máximo de generaciones.
    - Número de individuos por isla.
    - Número de individuos por enviar/recibir
    - Generaciones necesarias para ejecutar la migración.
- Algoritmo
    - Método de selección.
    - Probabilidad de selección.
    - Probabilidad de mutación.
- Problema
    - Problema a optimizar.
- Utilidades
    - Opción de guardar resultados.
    - Opción de guardar seguimiento.
    - Opción de guardar la población.
    - Opción de cargar una población creada con anterioridad.
    - Graficar resultados.

Además de estos parámetros, la biblioteca MPHStudio nos permite especificar topologías propias de conexión entre las islas, solo se requiere establecer el origen y el destino de cada una de las conexiones que se requieran.

En cuanto a las funciones de generar población, cruza y mutación, se tienen implementadas las funciones para problemas de optimización numérica. En cuanto a las funciones para optimización combinatoria se le da al usuario la posibilidad de que agregue sus propias funciones, esto debido a que cada problema tiene diferente forma de representación y diferente forma de realizar las operaciones.

MPHStudio tiene algunos problemas de optimización numérica y combinatoria implementados como ejemplo. Para implementar algún otro problema, la biblioteca cuenta con una herramienta que permite agregar el problema y su código respectivo.

La ventana interfaz es como se muestra en la siguiente imagen.

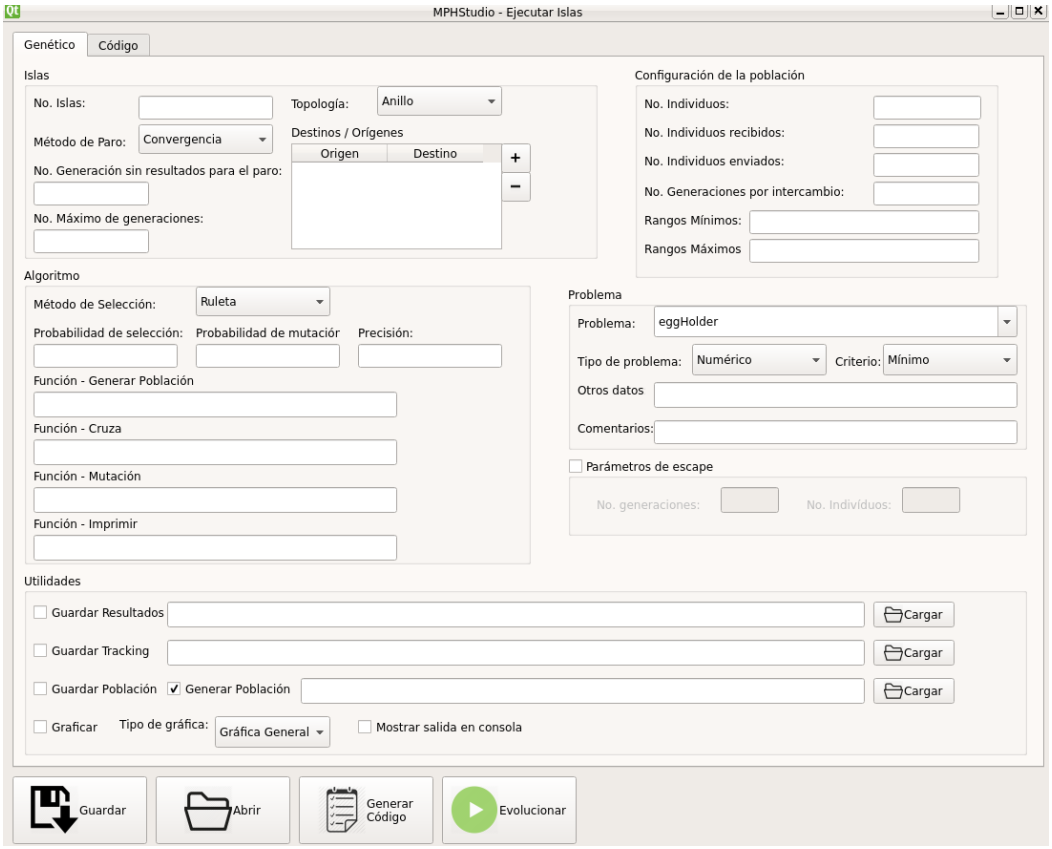


Figura 21 - Ventana de ejecución de modelo de islas

Con esta interfaz es posible establecer los parámetros de una forma sencilla y adaptada a las necesidades del usuario. La interfaz permite guardar/cargar un archivo con los parámetros ya establecidos.

Se tiene implementado solamente dos métodos de paro: el primero es el método de paro por número de iteraciones y el segundo es el método de paro por convergencia.

Las topologías con las que cuenta la biblioteca son: Anillo, estrella, todos conectados, y como se mencionó antes, es posible establecer una topología propia.

Si se requiere es posible generar el código fuente con los parámetros establecidos y poder ajustar a criterio propio o agregar más código simplemente presionando el botón *Generar Código.*

4.2.3 Interfaz del algoritmo genético celular

Al igual que la implementación de algoritmo genético distribuido, esta implementación requiere de una serie de parámetros que nos permitirá realizar diferentes experimentos que se ajusten más a nuestras necesidades.

Los parámetros requeridos son:

- Celular
    - Método de paro
    - Número de generaciones sin resultados para el paro
    - Número de generaciones máximo.
    - Forma del vecindario.
    - Radio del vecindario.
    - Número de columnas/filas de la malla de la población.
- Configuración de la población
    - Número de parejas por generación.
- Problema
    - Rango máximo/mínimo del problema (Solo aplica en problemas de optimización numérica).
    - Problema
    - Tipo de problema (Máximo o mínimo)
- Algoritmo
    - Método de selección.
    - Probabilidad de selección (Solo aplica para el método de selección por torneo)
    - Probabilidad de mutación.
- Utilidades
    - Comentarios
    - Opción de guardar resultados.
    - Opción de guardar seguimiento.

- Opción de guardar la población.
- Opción de cargar una población creada con anterioridad.
- Graficar resultados.

Todos estos parámetros son fácilmente establecidos para la ejecución de un experimento utilizando nuestra biblioteca MPHStudio.

La interfaz es como se muestra en la siguiente imagen:

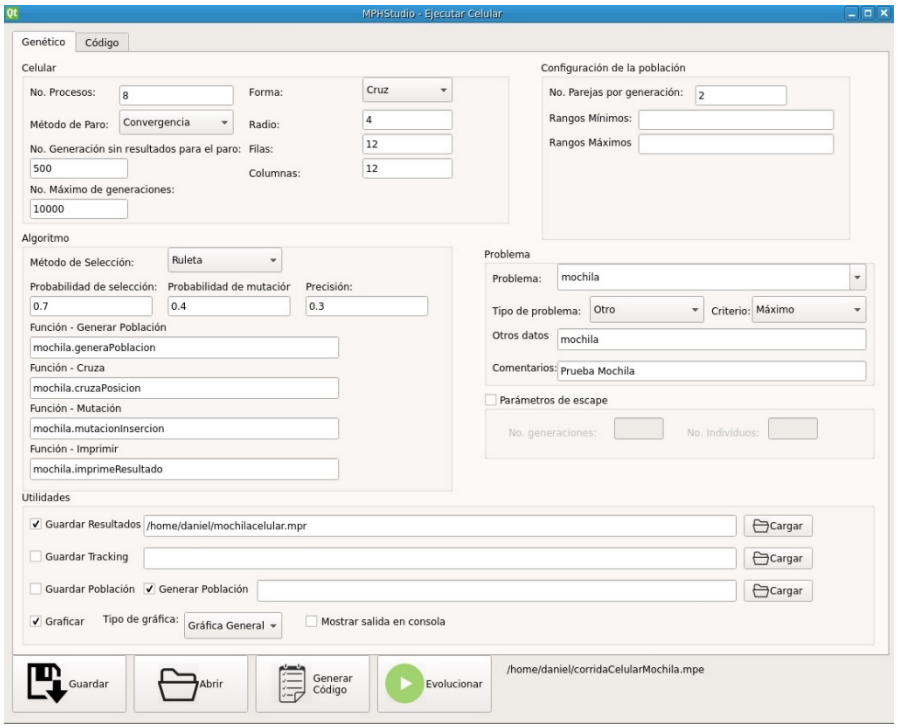


Figura 22 - Ventana de ejecución del modelo celular

De igual manera que la implementación de algoritmos genéticos distribuidos, la implementación de esta interfaz permite guardar la población inicial para que pueda ser utilizada en otro experimento, guardar archivo de parámetros, guardar los resultados de la ejecución de los experimentos y guardar un archivo de seguimiento que nos permita analizar posteriormente el comportamiento del algoritmo genético y los resultados que fue se fueron obteniendo durante todas las evoluciones.

Si se requiere, así como en la implementación del algoritmo genéticos distribuido, es posible agregar código en python dentro de esta biblioteca; para poder implementar algunas otras funciones o simplemente modificar el código que se generó con los parámetros establecidos.

La cantidad de procesos a utilizar en esta implementación se establece en la configuración de la biblioteca MPHStudio la cual se hablará más adelante.

La forma de mostrar los resultados dependerá de los parámetros que nosotros hayamos establecido, como por ejemplo:

- Resultados en consola.
- Resultados almacenados en un archivo.
- Gráfica de convergencia por cada una de las poblaciones o vecindarios en este caso.
- Gráfica general.

4.2.4 Herramientas de MPHStudio

La biblioteca MPHStudio cuenta con una serie de herramientas que nos permiten analizar los resultados obtenidos en los experimentos así como analizar el comportamiento de todo el algoritmo en cada evolución.

Para utilizar cada una de estas herramientas, solamente nos tenemos que dirigir al menú herramientas de la biblioteca y seleccionar la que deseemos.

Herramienta de análisis de resultados

La primera herramienta que explicaremos es la herramienta de análisis de resultados. Esta herramienta nos permite observar los resultados que se obtuvieron en la ejecución de cierto experimento, como son:

- Los valores óptimos
- La aptitud de los valores óptimos obtenidos.
- Total de generaciones que evolucionaron para llegar al resultado.
- Tiempo de ejecución.

- Gráfica de convergencia por población
- Gráfica de convergencia general.

Otros datos que son mostrados, son los parámetros con que fue ejecutado el experimento y comentarios si es que se establecieron previo a la ejecución del experimento.

También es posible guardar en un archivo de texto plano los valores óptimos que se obtuvieron en cada generación.

La interfaz de esta herramienta es como se muestra en la siguiente figura:

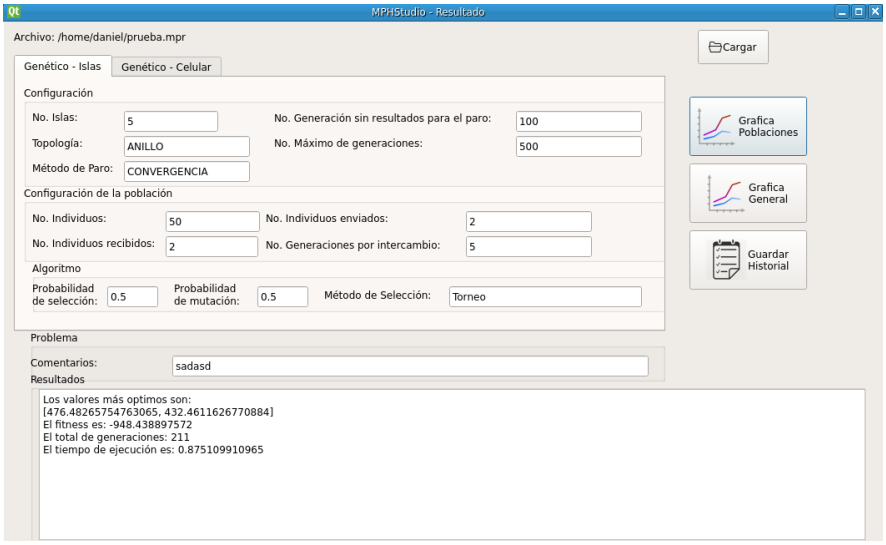


Figura 23 - Herramienta de análisis de resultados

Esta ventana permite cargar el archivo de resultados con la extensión *.mpr*. Al cargar el archivo la pestaña *Genético – Islas* mostrará los datos si es que se trata de un archivo de resultados de algoritmo genético distribuido, de otra forma, si se trata de un archivo de resultados de algoritmo genético celular, la pestaña *Genético – Celular* cargará los datos.

Ya cargados los datos se pueden mostrar las gráficas presionando el botón *Gráfica Poblaciones,* si deseamos mostrar la gráfica de todas las poblaciones que se utilizaron en el algoritmo genético, o

Gráfica General si se desea mostrar la gráfica de convergencia promedio de todas las poblaciones o vecindarios.

Al seleccionar gráfica por poblaciones, la biblioteca MPHStudio nos da la opción de poder seleccionar las poblaciones que deseamos en la gráfica.

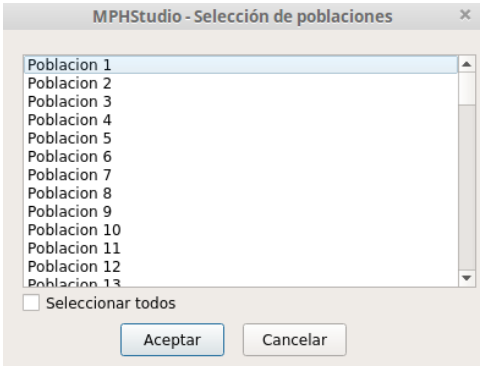


Figura 24 - Ventana de selección de poblaciones

Se puede seleccionar desde una población hasta todas.

Al seleccionar las que deseamos, la gráfica se mostrará como la siguiente imagen.

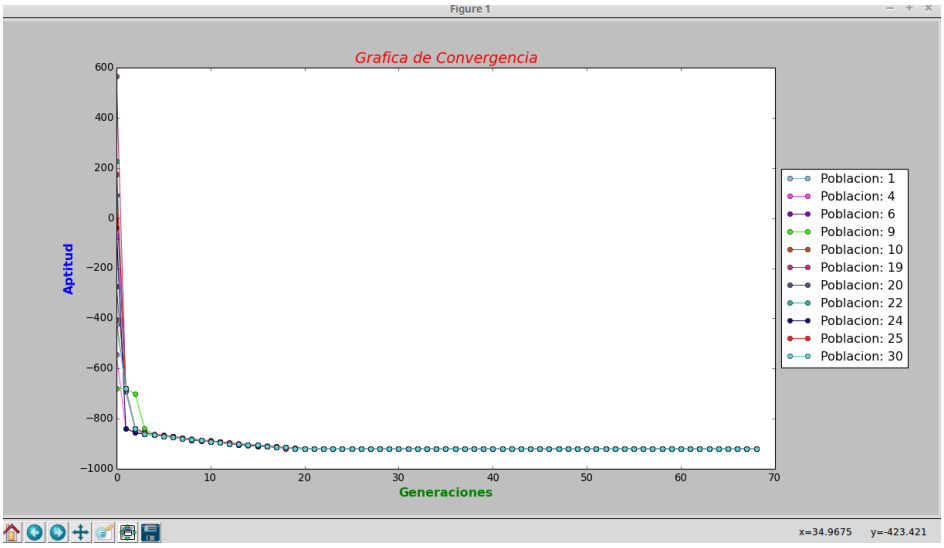


Figura 25 - Gráfica de convergencia

En esta ventana podemos desplazarnos en toda la gráfica para una observación más detallada así como un acercamiento o alejamiento de la imagen según deseemos, así mismo tenemos la opción de guardar la imagen de la gráfica en nuestra máquina con alguno de los formatos de imagen más utilizados.

Si por el contrario solo se desea mostrar la gráfica general de convergencia, aparecerá la ventana con la gráfica inmediatamente.

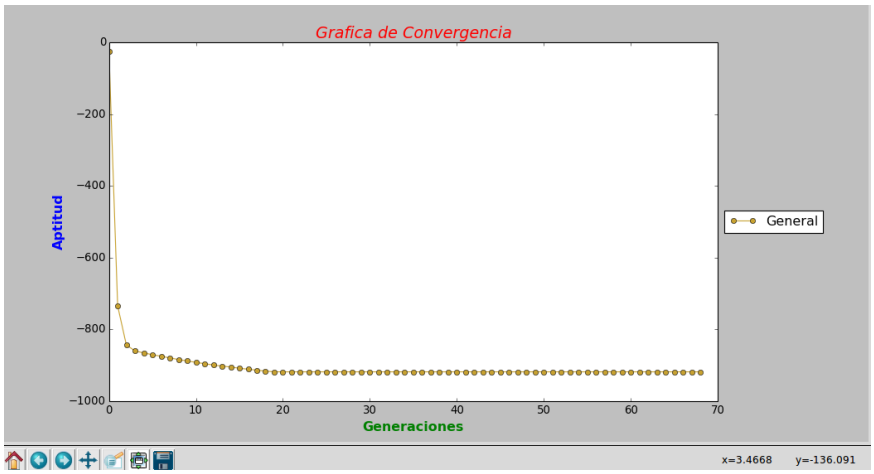
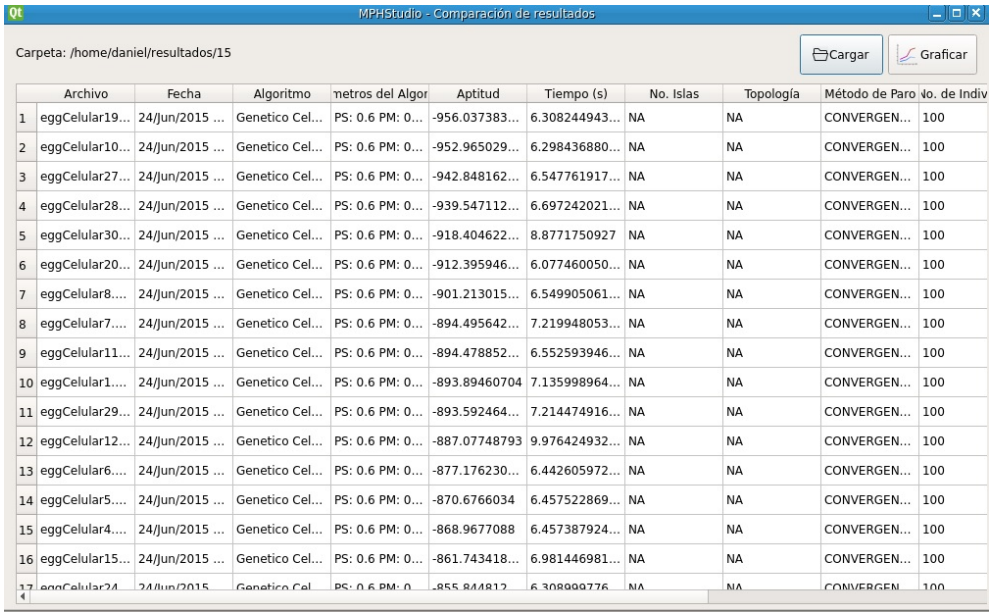


Figura 26 - Gráfica de convergencia promedio

Así como la gráfica de convergencia por poblaciones, esta ventana cuenta con las mismas opciones que se mencionaron antes.

Herramienta de bitácora de comparación de resultados

La herramienta de comparación de resultados, como su nombre lo indica, es una herramientas que nos permite seleccionar diferentes archivos de resultados que queramos y nos mostrará una tabla con los valores de resultados que se obtuvieron así como los algunos de los principales parámetros con que fueron ejecutados estos experimentos.



	Archivo	Fecha	Algoritmo	metros del Algor	Aptitud	Tiempo (s)	No. Islas	Topología	Método de Paro	No. de Indiv
1	eggCelular19...	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-956.037383...	6.308244943...	NA	NA	CONVERGEN...	100
2	eggCelular10...	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-952.965029...	6.298436880...	NA	NA	CONVERGEN...	100
3	eggCelular27...	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-942.848162...	6.547761917...	NA	NA	CONVERGEN...	100
4	eggCelular28...	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-939.547112...	6.697242021...	NA	NA	CONVERGEN...	100
5	eggCelular30...	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-918.404622...	8.8771750927	NA	NA	CONVERGEN...	100
6	eggCelular20...	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-912.395946...	6.077460050...	NA	NA	CONVERGEN...	100
7	eggCelular8....	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-901.213015...	6.549905061...	NA	NA	CONVERGEN...	100
8	eggCelular7....	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-894.495642...	7.219948053...	NA	NA	CONVERGEN...	100
9	eggCelular11...	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-894.478852...	6.552593946...	NA	NA	CONVERGEN...	100
10	eggCelular1....	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-893.89460704	7.135998964...	NA	NA	CONVERGEN...	100
11	eggCelular29...	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-893.592464...	7.214474916...	NA	NA	CONVERGEN...	100
12	eggCelular12...	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-887.07748793	9.976424932...	NA	NA	CONVERGEN...	100
13	eggCelular6....	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-877.176230...	6.442605972...	NA	NA	CONVERGEN...	100
14	eggCelular5....	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-870.6766034	6.457522869...	NA	NA	CONVERGEN...	100
15	eggCelular4....	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-868.9677088	6.457387924...	NA	NA	CONVERGEN...	100
16	eggCelular15...	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-861.743418...	6.981446981...	NA	NA	CONVERGEN...	100
17	eggCelular24...	24/Jun/2015 ...	Genetico Cel...	PS: 0.6 PM: 0...	-855.844812...	6.308999776...	NA	NA	CONVERGEN...	100

Figura 27 - Herramienta de comparación de resultados

Esta herramienta nos permite mostrar una gráfica de convergencia donde se puede comparar los valores promedio de las aptitudes de las poblaciones de cada uno de los archivos de resultados que se cargaron.

Con los datos cargados en la tabla, podemos obtener estadísticas como son:

- Tiempo promedio de ejecución
- Generaciones promedio
- Aptitud promedio
- Mejor y peor aptitud
- Mejor y peor tiempo de ejecución

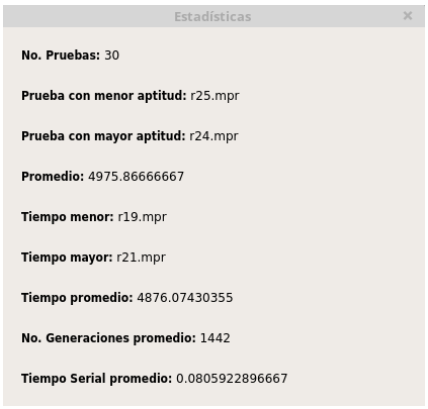


Figura 28 - Ventana de estadísticas

Herramienta de análisis del comportamiento del algoritmo

La herramienta de análisis del comportamiento del algoritmo genético nos permite observar y analizar los individuos que se van generando en cada uno de los pasos del algoritmo genético es decir, se registran los valores, la aptitud del individuo y a que población pertenece después de las funciones:

- Generación de la población
- Selección
- Cruza
- Mutación
- Operación de escape

En cuanto a la operación de selección, se registran los individuos que se seleccionaron para realizar la cruza.

La interfaz de esta herramienta es como se muestra en la siguiente figura.

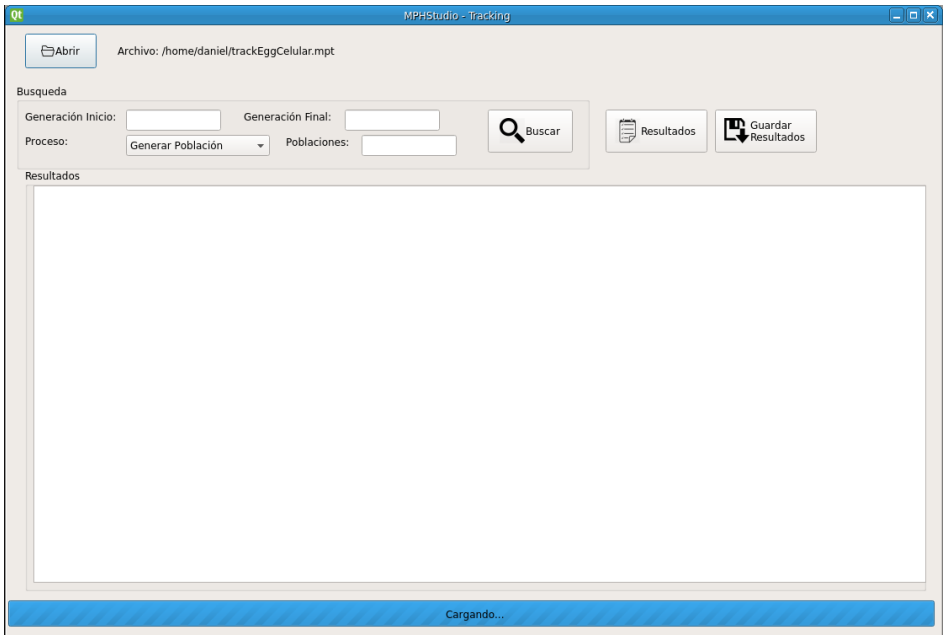


Figura 29 - Herramienta de seguimiento de ejecución

Para iniciar tenemos que cargar el archivo con la extensión *.mpt*, este es el tipo de archivo con el que se guardan los datos del seguimiento de la ejecución del algoritmo.

Las búsquedas nos permiten cargar los datos seleccionando la función del algoritmo genético que se quiera, el rango de generaciones y las poblaciones.

Si se necesita se puede ver los resultados obtenidos y los parámetros con que fue ejecutado el experimento, así como también es posible guardar el archivo de resultados en un archivo por separado.

Cabe mencionar que el archivo de seguimiento suele ser de un tamaño en MB muy grande debido a la gran cantidad de datos que se guardan. El tamaño del archivo va a depender de la cantidad de generaciones de evolución que le tardó al algoritmo encontrar el valor óptimo.

El archivo de seguimiento es un archivo que a su vez contiene otros archivos, cada archivo es un archivo de una función del algoritmo genético y una población o vecindario en el caso del algoritmo genético celular. Cuando el archivo se abre con la biblioteca MPHStudio, se copian los archivos mencionados con anterioridad en el directorio de archivos temporales del sistema.

En nuestra implementación del algoritmo genético distribuido, cada uno de los procesos que controlan a una de las islas del algoritmo, se encarga de registrar los datos de las operaciones del algoritmo genético; cuando se detiene la evolución de las poblaciones, es decir cuando se cumple la condición de paro, el proceso 0 se encarga de empaquetar los archivos que cada uno de los procesos crearon junto con el archivo de resultados para al final general el archivo con extensión *.mpt*. Para el caso de la implementación del algoritmo genético celular, el proceso principal es el que se encarga de crear todos los archivos con los registros de las operaciones del algoritmo genético así como el empaquetado de todos estos.

Herramienta de múltiples ejecuciones

La herramienta de múltiples ejecuciones nos permite cargar un archivo de configuración de extensión *.mpe* y poder ejecutar N número de veces el algoritmo que hayamos configurado y poder guardar los resultados en N número archivos de resultados. Esta herramienta nos permite obtener estadísticas con los parámetros que se establezcan del algoritmo genético.

Con esta herramienta también podemos ejecutar nuestras pruebas secuenciales con la finalidad de realizar un análisis de la aceleración en nuestros resultados.

La interfaz de la herramienta es como la que se muestra en la siguiente imagen.

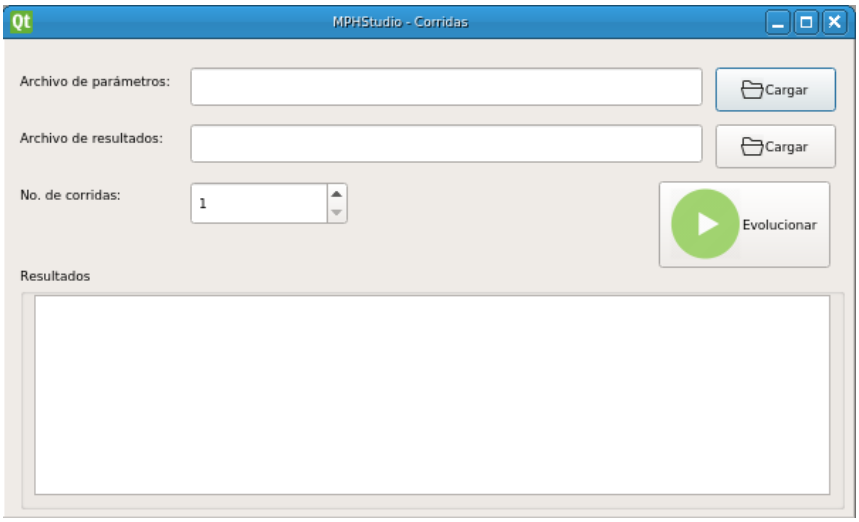


Figura 30 - Herramienta de múltiples ejecuciones

Capítulo 5

PRUEBAS Y RESULTADOS

Para este trabajo de investigación, se ocuparon algunas funciones de optimización numéricas así como un problema de optimización combinatoria. La función de optimización numéricas se tomó del CEC 2013 [45]. En cuanto a los problemas de optimización combinatoria se utilizaron los problemas del agente viajero (TSP) y problema de la mochila.

Para las pruebas utilizamos valores empíricos pero siguiendo los consejos de [14]. Estos valores se fueron modificando para poder estudiar el comportamiento de las implementaciones.

Las pruebas fueron realizadas utilizando un equipo con las siguientes características:

- **CPU:** Intel Core i7-3770 CPU @ 3.40 GHZ x 4
- **Memoria:** 8 GB
- **Sistema Operativo:** Linux Mint 17.1 64-bits

Los valores de los resultados se puede ver en las tablas del apéndice A.1.

Experimento 1. Función Ackley

En este experimento se quiere analizar la propuesta de paralelización de los algoritmos genéticos distribuido y celular, contra la implementación secuencial de estos mismos algoritmos utilizando una función de optimización numérica.

La función numérica que se utilizará para este experimento es la función Ackley. Esta función está definida como:

$$f(x,y) = -20\exp\left(-0.2\sqrt{0.5(x^2+y^2)}\right) - exp\big(0.5(\cos(2\pi x)+\cos(2\pi y))\big) + e + 20$$

Dominio: $-5 \leq x, y \leq 5$

Mínimo: $f(0,0) = 0$

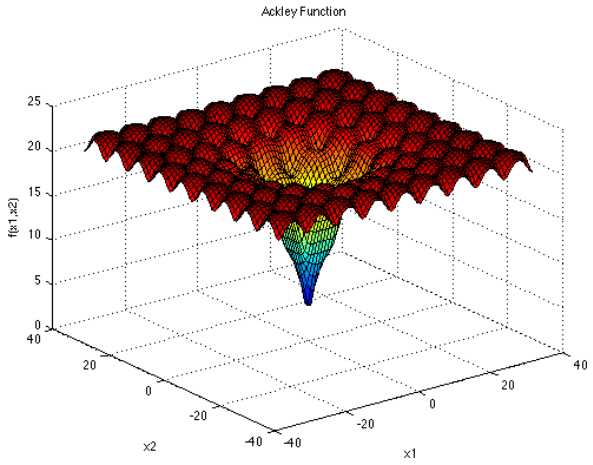


Figura 31 - Función Ackley

Para este experimento consideramos un óptimo f*(x,y) <0.000.

Escenarios 1 y 2. AGD Varias Poblaciones – Secuencial y Paralelo

Parámetros	Valores
Número de poblaciones	1,5,8,10
Topología	Anillo
Criterio de paro	Convergencia
Máximo número de generaciones sin cambio	50
Máximo número de generaciones	5000
Número de individuos por población	50
Número de migrantes	5
Frecuencia de migración	10
Técnica de selección	Ruleta
Probabilidad de selección	0.2
Varianza en la mutación	0.05

Tabla 4 - Tabla de parámetros para la ejecución del experimento 1 - escenarios 1 y 2

Resultados

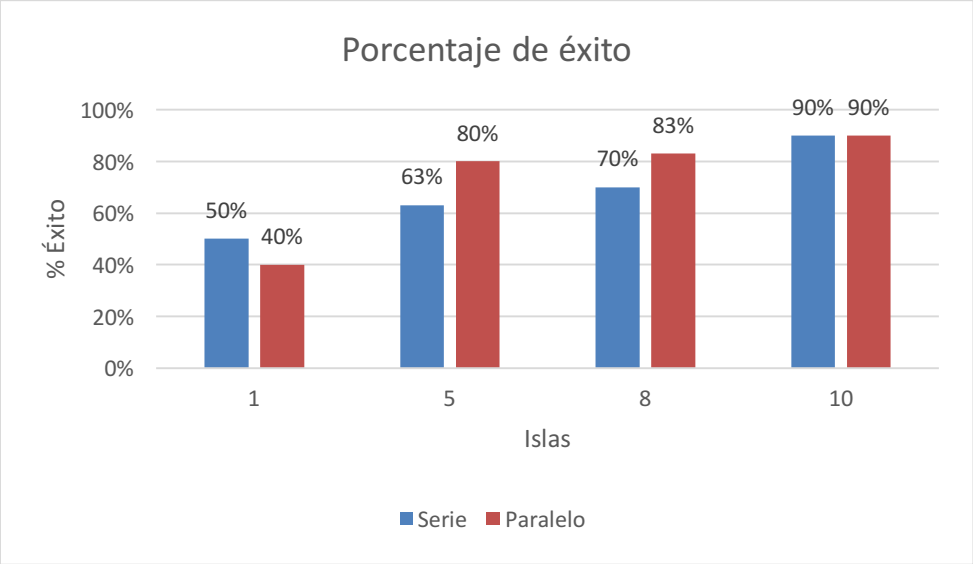


Figura 32 - Gráfica del porcentaje de ejecuciones que obtuvieron el óptimo experimento 1 – escenarios 1 y 2

En la gráfica anterior se puede observar que al aumentar el número de poblaciones (islas), el porcentaje de éxito aumenta. La hipótesis que se tiene de esto es que debido al aumento en el número de las poblaciones, se tiene una mayor diversidad de individuos en el espacio de búsqueda teniendo de esta forma una mayor exploración.

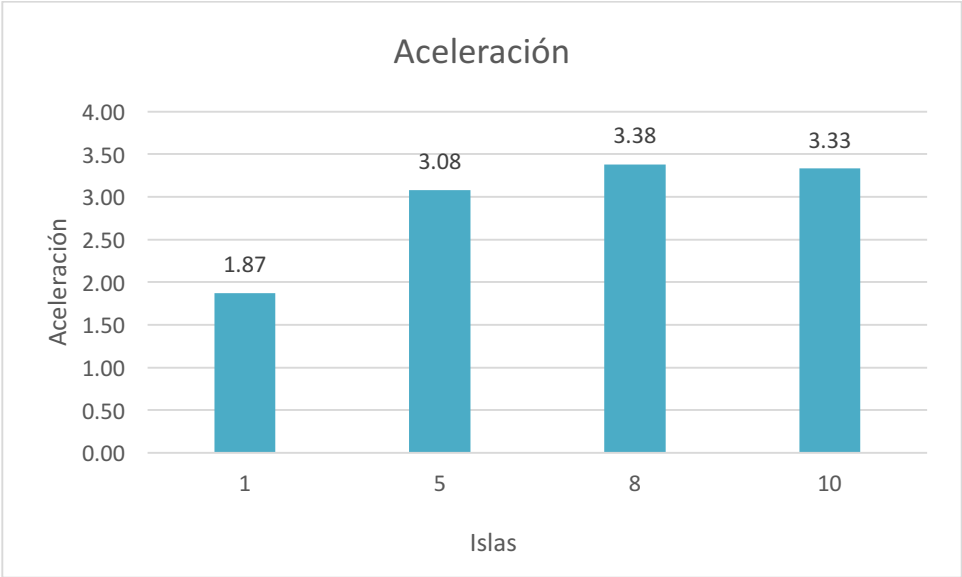


Figura 33 - Gráfica de aceleración del experimento 1 – escenarios 1 y 2

En la gráfica de aceleración anterior, se observa que al aumentar el número de islas aumenta la aceleración, esto debido a que al aumentar las islas aumenta el número de unidades de procesamiento. Se puede ver que se llega a un punto en que la aceleración empieza a bajar debido a que al aumentar las islas se requiere más tiempo en procesar la migración y a la sincronización de las unidades de procesamiento.

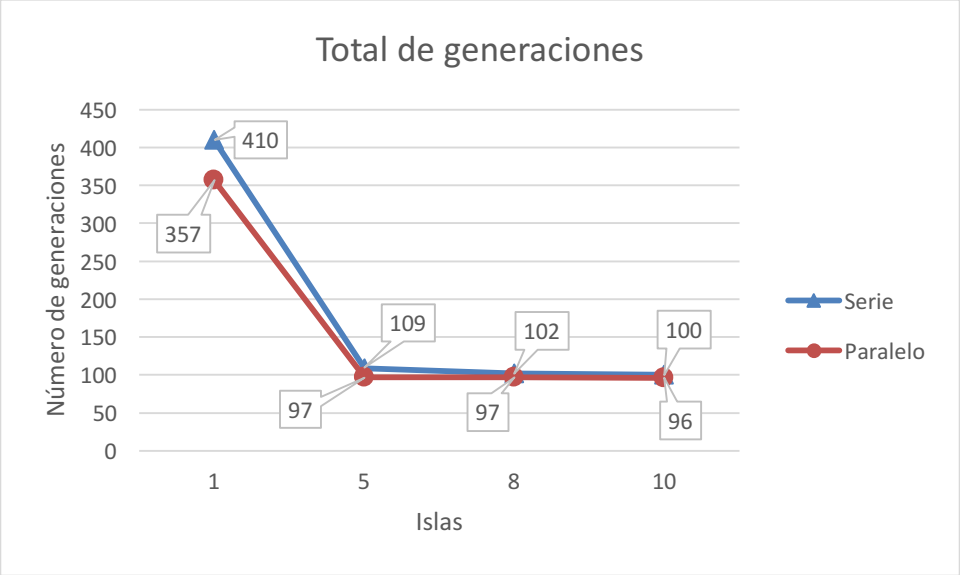


Figura 34 - Gráfica del total de generaciones que le tomó al algoritmo obtener el óptimo del experimento 1 – escenarios 1 y 2

Al tener una única población de individuos, al algoritmo le toma más tiempo poder encontrar el óptimo debido a que la búsqueda puede caer en un óptimo local. En la gráfica anterior se puede observar que con una sola isla, el algoritmo necesitó de un mayor número de generaciones para poder encontrar una buena solución. Al requerir un mayor número de generaciones, el tiempo de ejecución aumentó; esto se puede ver en la gráfica siguiente. Además en esta gráfica se observa que al aumentar el número de islas aumenta el tiempo de ejecución, siendo en la implementación secuencial más radical el aumento del tiempo.

Figura 35 - Gráfica del tiempo de ejecución del experimento 1 - escenarios 1 y 2

Escenarios 3 y 4. AGC Variación del radio del vecindario – Secuencial y Paralelo

Parámetros	Valores
Tamaño de la malla	35x35
Forma	Cruz
Radio	1,3,6,10,13,15
Tipo	Síncrono
Procesos	1,8
Criterio de paro	Convergencia
Máximo número de generaciones sin cambio	150
Máximo número de generaciones	5000
Técnica de selección	Ruleta
Probabilidad de selección	0.5
Probabilidad de mutación	0.5

Tabla 5 - Tabla de parámetros para la ejecución del experimento 1 - escenarios 3 y 4

Resultados

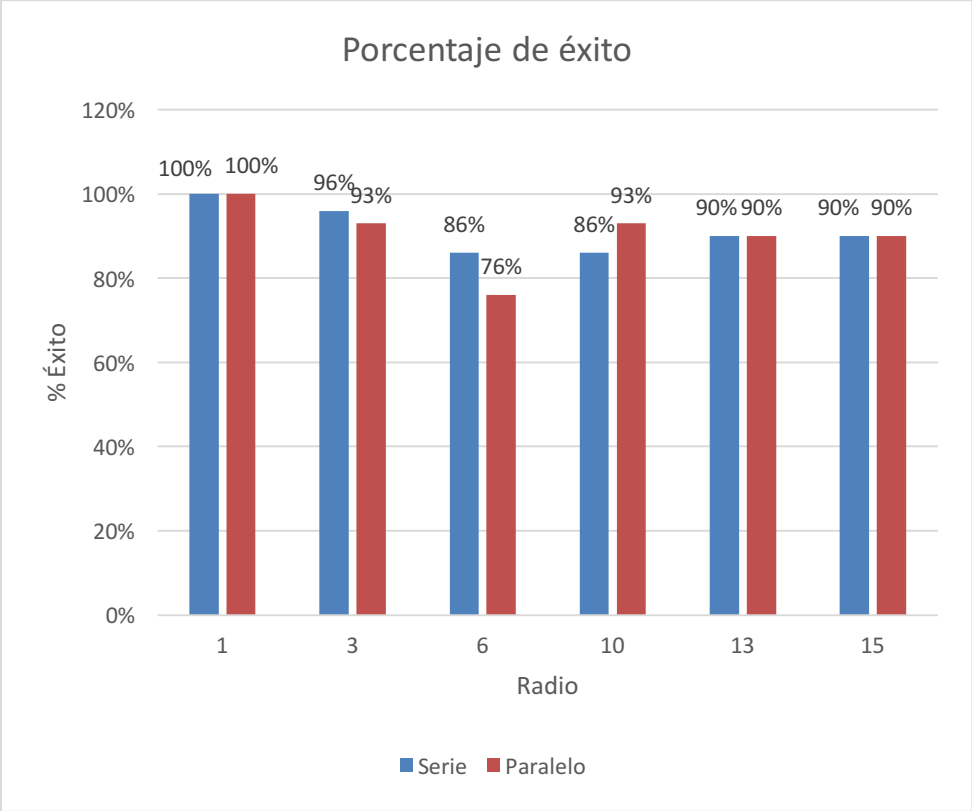


Figura 36 - Gráfica del porcentaje de ejecuciones que obtuvieron el óptimo experimento 1 – escenarios 3 y 4

Los mejores resultados en cuanto al porcentaje de éxito, son cuando el radio de la vecindad es 1, esto debido a que al tener un radio pequeño existe poca interacción entre los vecindarios provocando que exista poca migración de individuos. Una de las características principales del algoritmo celular es que su convergencia es demasiado rápida, especialmente cuando el radio de los vecindarios es grande. Esta convergencia tan rápida puede provocar que el algoritmo se atore en un óptimo local.

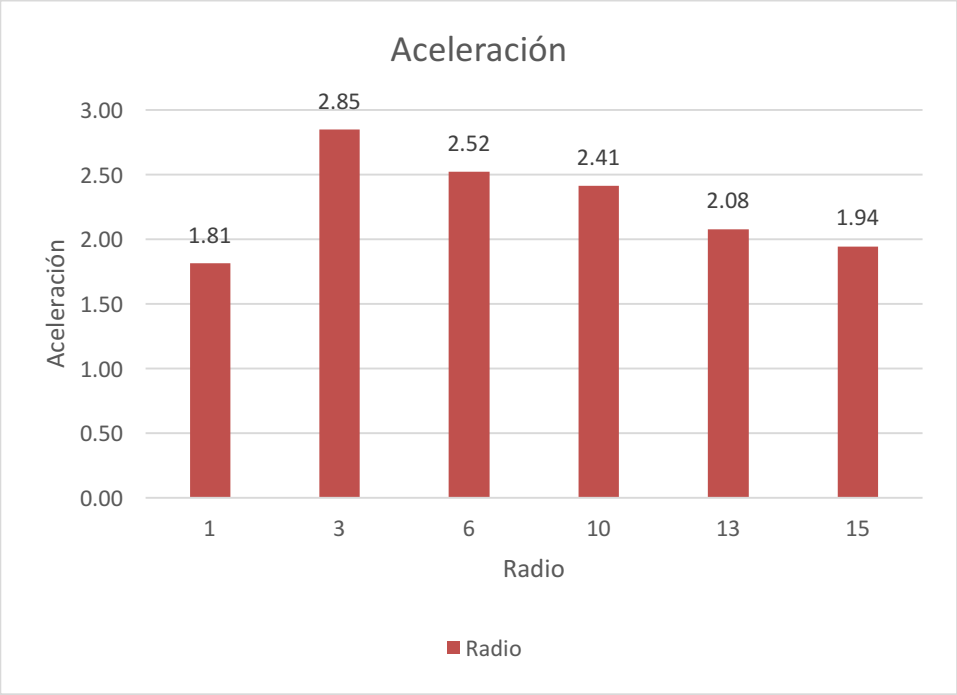


Figura 37 - Gráfica de aceleración del experimento 1 – escenarios 3 y 4

En la gráfica anterior, se puede observar que la mayor aceleración se obtuvo en la ejecución del algoritmo con un radio del vecindario de valor 3, esto se debió a que el tiempo de ejecución secuencial fue muy grande en comparación con el tiempo de ejecución en paralelo, lo cual se puede apreciar en la figura 38.

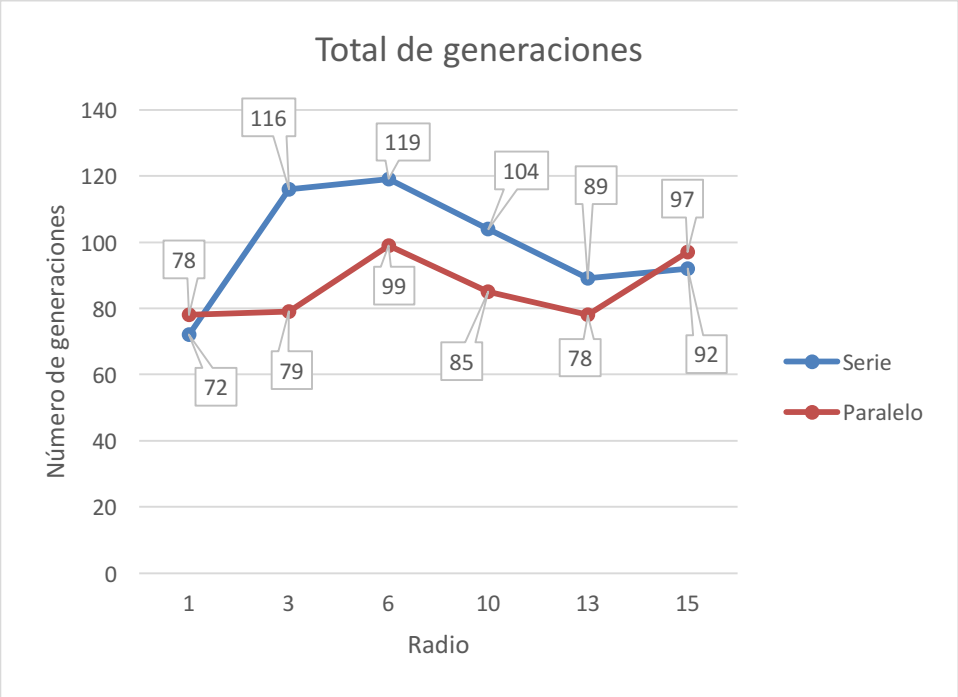


Figura 38 - Gráfica del total de generaciones que le tomó al algoritmo obtener el óptimo del experimento 1 – escenarios 3 y 4

En la gráfica anterior se observa un aumento en el número de generaciones cuando el radio del vecindario es 3 y 6. Este aumento está completamente relacionado con el tiempo de ejecución, es decir, al aumentar el número de generaciones el tiempo de ejecución aumenta.

Cabe mencionar que el aumentar el radio de la vecindad no necesariamente aumenta el tiempo de ejecución. Esto se puede apreciar en la gráfica siguiente. La línea que corresponde a la ejecución en paralelo se muestra casi lineal debido a que el número de generaciones que le tomó al algoritmo llegar al óptimo fue casi igual en todas las pruebas.

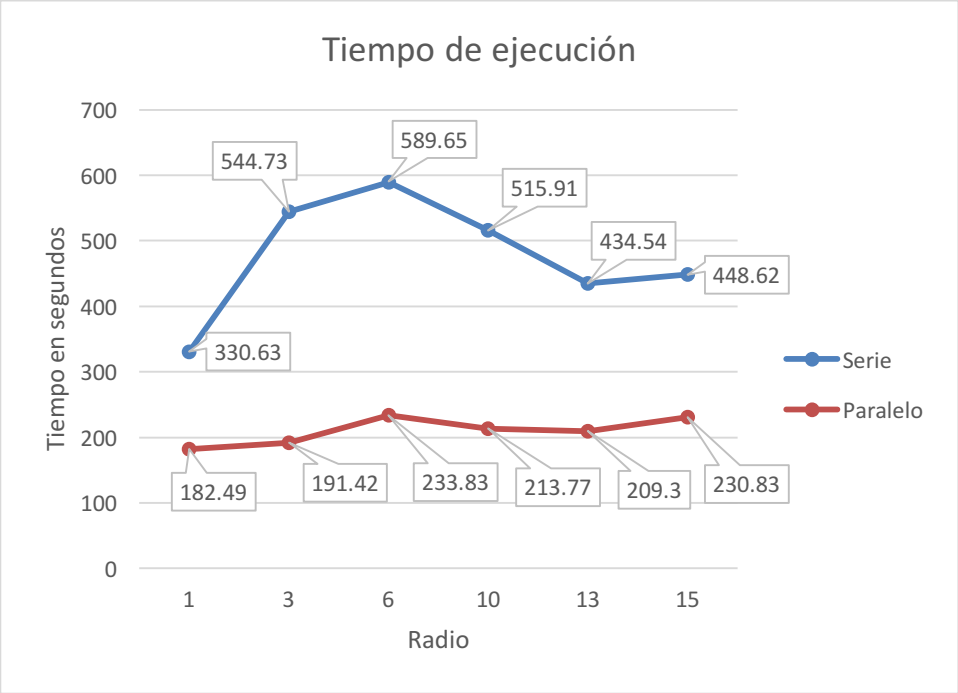


Figura 39 - Gráfica del tiempo de ejecución del experimento 1 - escenarios 3 y 4

El tiempo de ejecución en paralelo, en todos los radios, siempre fue mejor que el tiempo secuencial. Esto muestra que la implementación en paralelo propuesta, mejora el tiempo de ejecución y en la mayoría de los casos mejora la calidad de las soluciones.

Experimento 2. Problema del agente viajero

El problema del agente viajero (TSP - *Traveling Salesman Problem*) se relaciona con el problema de encontrar un ciclo hamiltoniano en un grafo. El problema es: Dado un grafo ponderado G, encuentre en G un ciclo de Hamilton con longitud mínima. Si se piensa en los vértices de un grafo ponderado como ciudades y en los pesos de las aristas como distancias, el problema del agente viajero consiste en encontrar una ruta más corta en la que el agente viajero pueda visitar cada ciudad una vez, comenzando y terminando en la misma ciudad [46].

Todos los algoritmos para encontrar ciclos de Hamilton requieren un tiempo ya sea exponencial o factorial en el peor de los casos.

En este experimento queremos analizar la propuesta de paralelización del algoritmo genético distribuido, contra la implementación secuencial de este mismo algoritmos utilizando una función de optimización combinatoria.

Para los experimentos se utilizó el conjunto de datos wi29.tsp. Este conjunto de datos es un conjunto de ciudades de Western Sahara que consta de 29 ciudades. El conjunto de datos fue obtenido del sitio de la Universidad de Waterloo en Ontario, Canadá. El sitio es www.math.uwaterloo.ca/tsp/data/

Para este experimento se consideró un óptimo f*(x) <27,700. El óptimo tiene un valor de aptitud = 27603

Escenario 1 y 2. AGD Varias poblaciones – Secuencial y Paralelo

Parámetros	Valores
Número de poblaciones	5,8,10,15
Topología	Anillo
Criterio de paro	Convergencia
Máximo número de generaciones sin cambio	1000
Máximo número de generaciones	10,000
Número de individuos por población	300
Número de migrantes	1
Frecuencia de migración	20
Técnica de selección	Ruleta
Función de cruza	Cruza ordenada
Función de mutación	Mutación por inserción
Probabilidad de selección	0.5
Probabilidad de mutación	0.5
Función de escape	
Número de generaciones	100
Número de individuos a reemplazar	280

Tabla 6 - Tabla de parámetros para la ejecución del experimento 2 - escenarios 1 y 2

Debido a la complejidad del problema, se decidió utilizar la función de escape que nos permite tener una mayor exploración en el espacio de búsqueda, ya que es muy susceptible a quedar atorado en un óptimo local.

Aunque se tengan las funciones de cruza y mutación, estas no son suficientes para este problema, ya que un pequeño cambió en el genotipo del individuo, provoca un gran cambio en el fenotipo de este; tal vez siendo mejor el cambio pero casi siempre, en la mayoría de los casos peor.

Resultados

Con los parámetros propuestos, se obtuvieron muy buenos resultados en cuanto a la calidad de la soluciones y a la cantidad de ejecuciones que obtuvieron el óptimo.

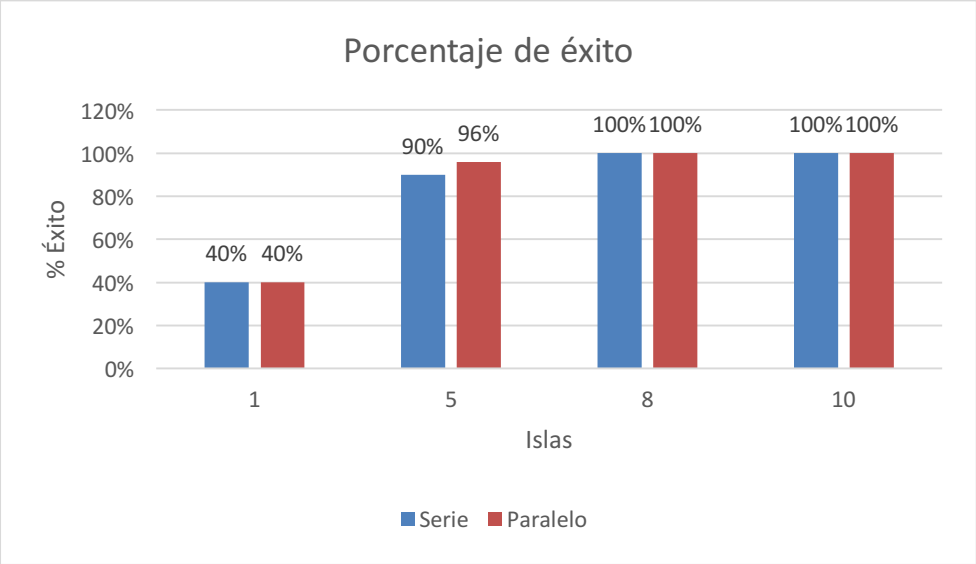


Figura 40 - Gráfica del porcentaje de ejecuciones que obtuvieron el óptimo experimento 2 – escenarios 1 y 2

La gráfica anterior muestra el porcentaje de ejecuciones que pudieron obtener el óptimo. El porcentaje más bajo resultó ser en la ejecución con una sola población, esto debido a que como se mencionó con anterioridad, es muy baja la diversidad de individuos dentro del espacio de búsqueda, provocando que la búsqueda se quede atorada en un óptimo local.

Conforme se fueron aumentando el número de islas, el porcentaje de éxito mejoró considerablemente, llegando a obtenerse hasta el 100% tanto en implementaciones secuenciales y en paralelo.

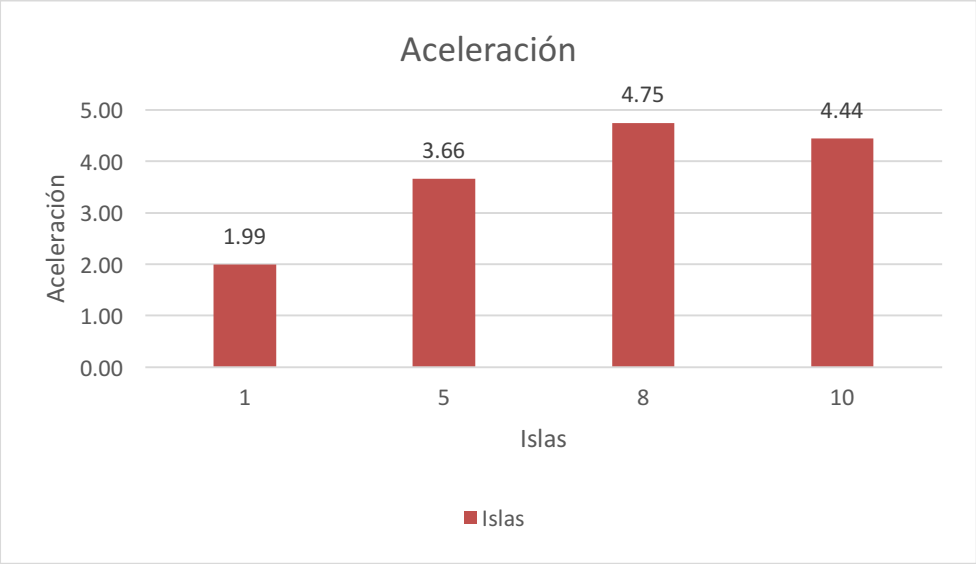


Figura 41 - Gráfica de aceleración del experimento 2 – escenarios 1 y 2

En este experimento la mejor aceleración obtenida es cuando el número de islas es igual a 8, esto debido a que el equipo donde se ejecutaron las pruebas cuenta con 8 núcleos en el microprocesador. Conforme se va aumentando el número de islas mejora la aceleración, pero después la aceleración disminuye.

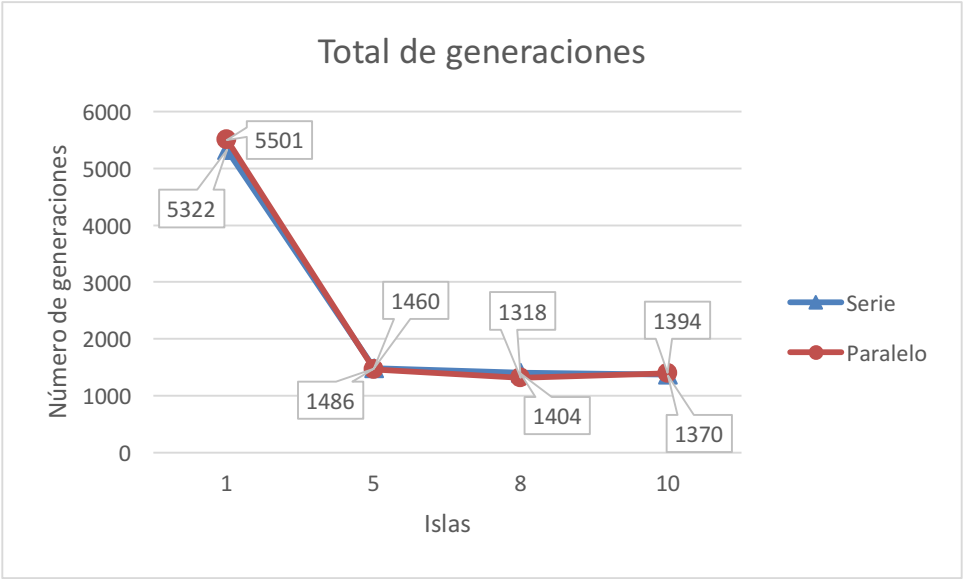


Figura 42 - Gráfica del total de generaciones que le tomó al algoritmo obtener el óptimo del experimento 2 – escenarios 1 y 2

En la gráfica anterior se puede apreciar que en la ejecución con una isla el número de generaciones que le tomó al algoritmo obtener el óptimo local fue mucho mayor que en los demás, esto debido a la falta de otras poblaciones, ya que al tener más poblaciones evolucionando se genera una mayor diversidad en la búsqueda, por lo tanto cuando solo se ejecuta una sola isla se puede llegar a presentar que la búsqueda se atore en un óptimo local.

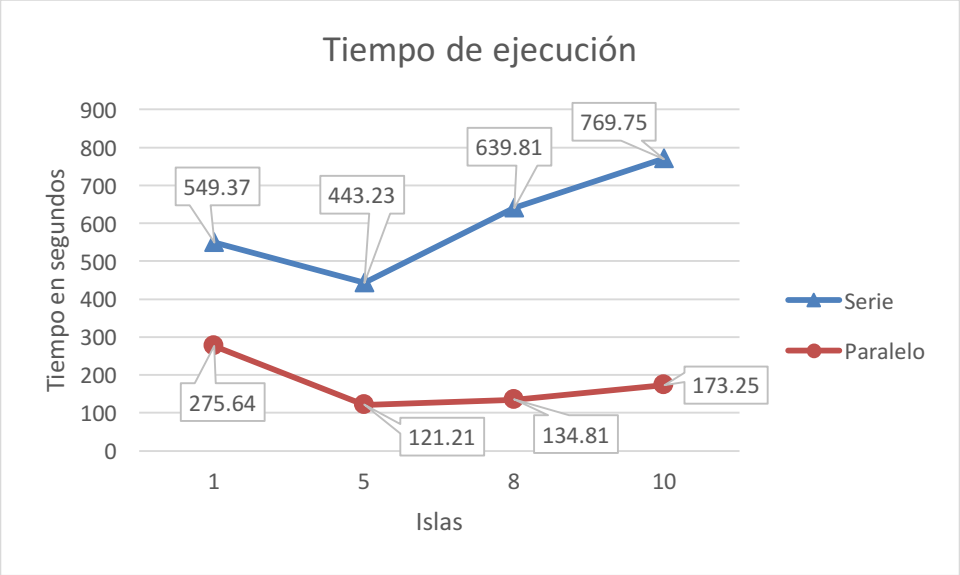


Figura 43 - Gráfica del tiempo de ejecución del experimento 2 - escenarios 1 y 2

El tiempo de ejecución en la ejecución secuencial; conforme se iba aumentado el número de islas, crecía de forma potencial, mientras que la ejecución en paralelo crecía de forma lineal.

Experimento 3. Problema de la mochila

El problema de la mochila es definido formalmente como: Se tiene un conjunto de objetos *N*, que consiste de *n* objetos *j* con ganancia *pj* y peso *wj*, y una capacidad *c*. (Usualmente, los valores toman números enteros positivos). El objetivo es seleccionar un subconjunto de *N* tal que la ganancia total de esos objetos seleccionados es maximizado y el total de los pesos no excede a *c*.

El problema de la mochila forma parte de una lista histórica de problemas NP − Completos elaborada por Richard Karp en 1972 [47].

En este experimento queremos analizar la propuesta de paralelización del algoritmo genético celular asíncrono contra el algoritmo genético celular síncrono con la finalidad de poder determinar las ventajas y desventajas que cada uno de ellos tiene.

Para los experimentos, se generó un conjunto de datos de 70 artículos con un peso máximo de 10,000. Este conjunto de datos fue generado utilizando un generador creado por investigadores del departamento de ciencias de la computación de la Universidad de Princeton en los Estados Unidos de América. El sitio es http://introcs.cs.princeton.edu/

Escenario 1 y 2. AGC Variación del número de procesos – Síncrono y Asíncrono

Parámetros	Valores
Tamaño de la malla	15x15
Forma	Cruz
Radio	1
Tipo	Síncrono, Asíncrono
Procesos	1,4,6,8
Criterio de paro	Convergencia
Máximo número de generaciones sin cambio	800
Máximo número de generaciones	10000
Técnica de selección	Ruleta
Probabilidad de selección	0.5
Probabilidad de mutación	0.5
Función de escape	
Número de generaciones	6
Número de individuos a reemplazar	223

Tabla 7 - Tabla de parámetros para la ejecución del experimento 3 - escenario 1

Resultados

Los resultados obtenidos en este experimento, nos permiten observar el comportamiento del algoritmo genético celular síncrono y asíncrono.

El porcentaje de éxito es similar en todos los casos ya que el variar el número de procesos no cambia el comportamiento del algoritmo.

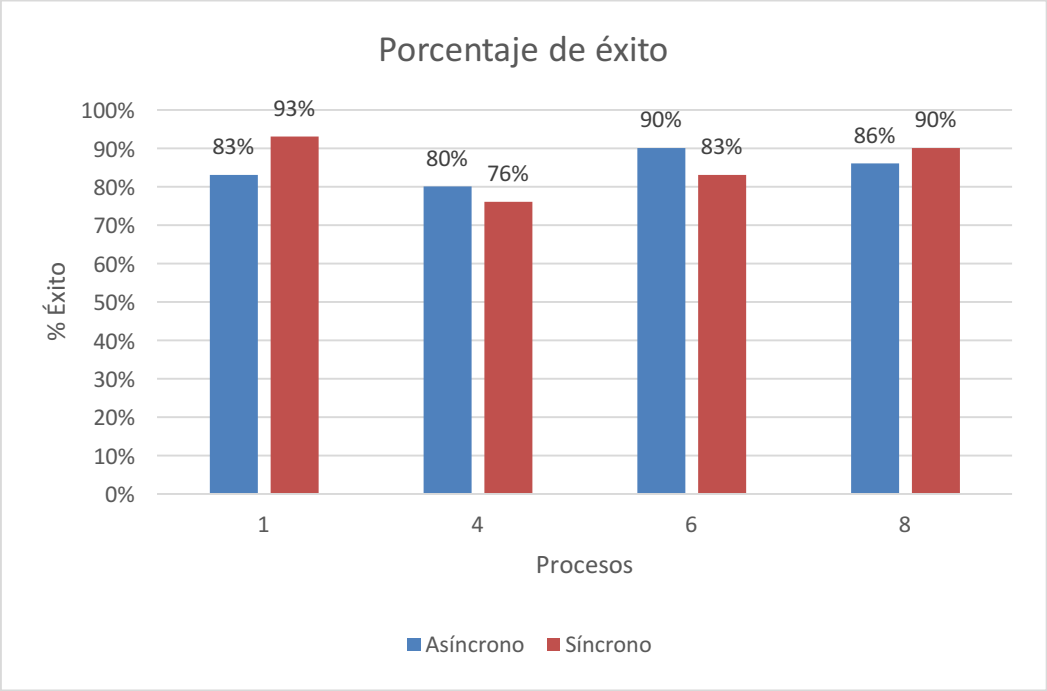


Figura 44 - Gráfica del porcentaje de ejecuciones que obtuvieron el óptimo experimento 3 – escenarios 1 y 2

En la siguiente gráfica podemos ver que la mejor aceleración la tiene el modelo asíncrono en la mayoría de los casos además se puede observar que la aceleración va aumentando conforme el número de procesos aumenta, pero en cierto punto la aceleración empieza a caer debido a que se tiene una limitante en la cantidad de procesador, es decir, las pruebas fueron realizadas en un equipo que tiene un procesador i7 de Intel que cuenta con 4 núcleos físicos y 4 núcleos virtuales.

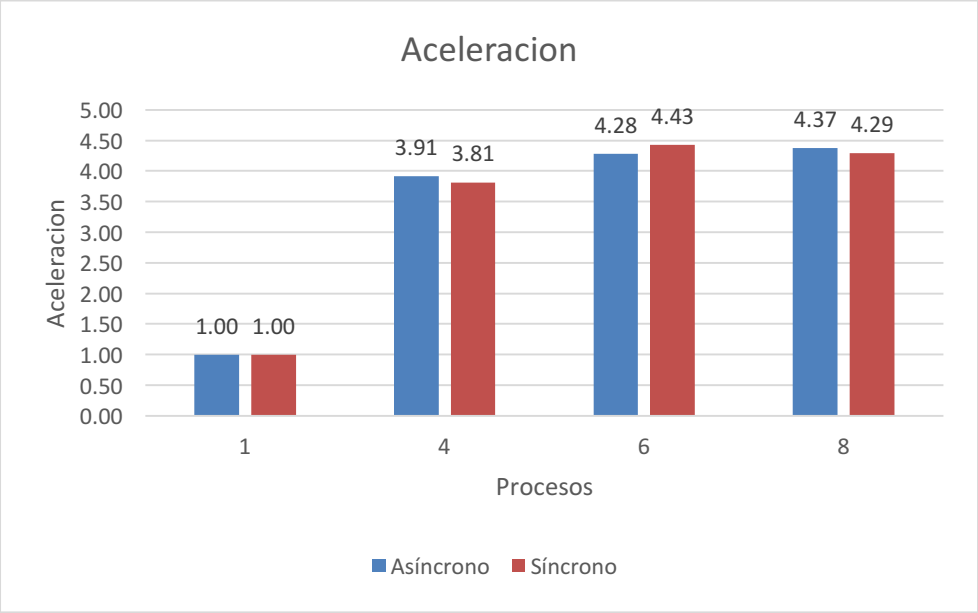


Figura 45 - Gráfica de aceleración del experimento 3– escenarios 1 y 2

En cuanto al número de generaciones, a la implementación síncrona le tomó menos generaciones en converger casi en todas las pruebas.

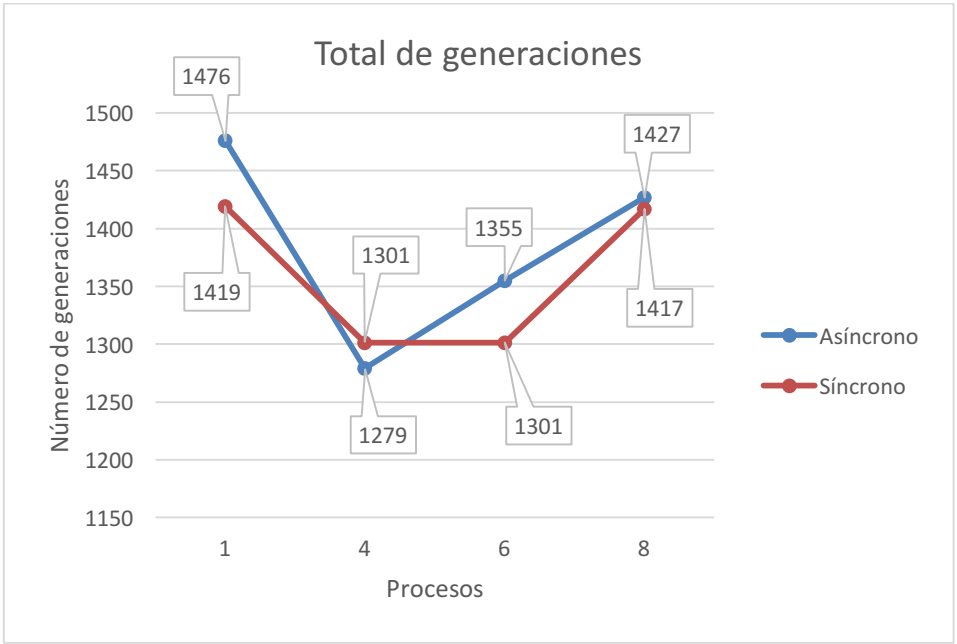


Figura 46 - Gráfica del total de generaciones que le tomó al algoritmo obtener el óptimo del experimento 3 – escenarios 1 y 2

El tiempo de ejecución utilizando un solo proceso es claramente superior que utilizando 4,6 y 8 procesos debido a que la ejecución no se realiza de forma concurrente. Esto se puede observar en la siguiente figura.

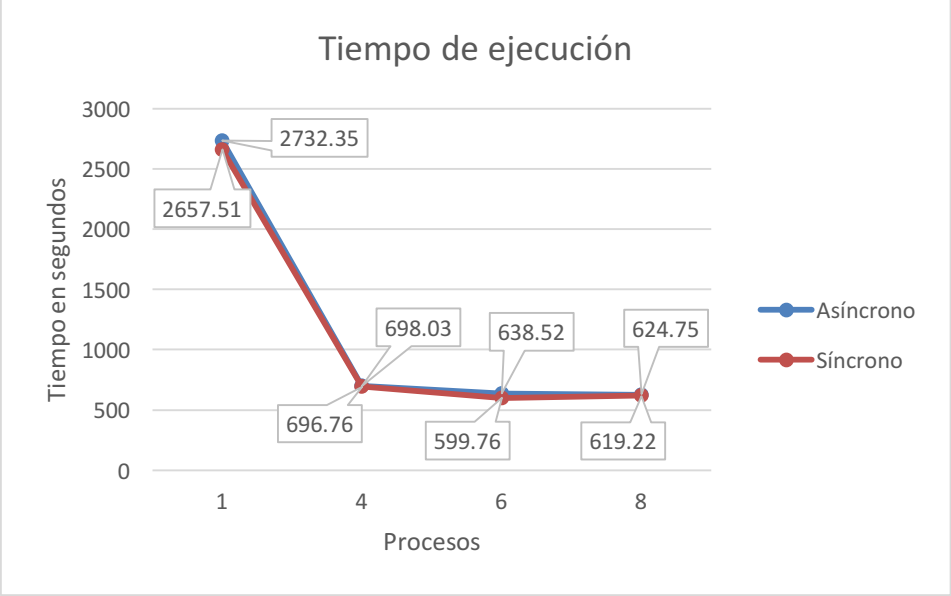


Figura 47 - Gráfica del tiempo de ejecución del experimento 3 - escenarios 1 y 2

Evaluación

Para la evaluación de nuestra propuestas de paralelización de los modelos de metaheurísticas multipoblacionales y de la biblioteca MPHStudio, se trató de comparar estas propuestas con las propuestas que están implementadas en las bibliotecas que se han investigado en el estado del arte.

Como se menciona en el capítulo 2, algunas bibliotecas en su documentación mencionan que pueden trabajar con el modelo de islas debido a que están paralelizados los métodos de las metaheurísticas, pero es necesario implementar la política de migración, lo cual conlleva a proponer un modelo propio para este tipo de metaheurísticas multipoblacionales.

En cuanto al modelo multipoblacional celular, casi ninguna biblioteca maneja estos modelos, con excepción de la biblioteca Mallba, pero el problema recae de nuevo en que es necesario establecer un modelo celular, ya que la biblioteca está preparada para trabajar con metaheurísticas en paralelo pero es por cuenta del usuario implementar los modelos multipoblacionales.

La siguiente tabla se muestra la comparativa de las bibliotecas mencionada en el Capítulo 2 y la biblioteca desarrollada para este trabajo.

Biblioteca	Modelo distribuido	Modelo celular asíncrono	Modelo celular síncrono	Interfaz gráfica	Herramientas de análisis
MPHStudio	✓	✓	✓	✓	✓
Global Optimization Toolbox for Matlab	✓	✗	✗	✓	✓
Mallba	?	?	?	✗	✗
Inspyred	?	✗	✗	✗	✗
Watchmaker	✓	✗	✗	✓	✗
PyGMO	✓	✗	✗	✗	✓

✓ Función implementada.

✗ Función no implementada.

? Se requiere implementar modelo propio.

Tabla 8 - Tabla comparativa de bibliotecas existentes y la propuesta en este trabajo.

Como se observa en la tabla anterior, las bibliotecas disponibles para poder realizar la evaluación son las bibliotecas *Watchmaker*, *PyGMO* y *Global Optimization Toolbox for Matlab*. La herramienta *Global Optimization Toolbox for Matlab* cuanta con el problema de que además de requerir el programa de *Matlab*, es necesario realizar la compra de este *toolbox* y el de paralelización, por lo cual se descarta la evaluación con esta herramienta.

Con la biblioteca *PyGMO* se trató de realizar pruebas utilizando el modelo distribuido, pero la instalación de esta biblioteca se complica con la compilación y la búsqueda de bibliotecas que requiere para su instalación, por lo que no se pudo instalar esta biblioteca en el mismo entorno donde se realizaron las pruebas utilizando la biblioteca MPHStudio.

Por último, el *framework Watchmaker* permite también la implementación del modelo distribuido según su documentación, el problema que se tiene con esta biblioteca es que los ejemplos mostrados no muestran la forma de utilizar el modelo de islas y su soporte se dejó de hacer desde el 2010.

Capítulo 6

CONCLUSIONES

Con base en los resultados obtenidos se puede concluir que en el modelo distribuido (islas) al aumentar el número de poblaciones se mejora la diversidad de la población haciendo que existan un mayor número de posibles soluciones en todo el espacio de búsqueda dando lugar a mejores resultados.

Al aumentar el número de poblaciones la aceleración va aumentando, pero llega un punto en el cual disminuye, esto debido a que el número de poblaciones más el proceso principal, sobrepasa el número de unidades de procesamiento, provocando una baja eficiencia en la implementación.

En los resultados también se puede observar claramente que siempre es mejor el uso de varias poblaciones para mejorar la calidad de las soluciones.

Para la implementación del modelo celular, se concluye que al aumentar el radio del vecindario, la calidad de las soluciones disminuye, esto debido al comportamiento del modelo celular, es decir, una de las principales características del modelo es que converge demasiado rápido y por lo tanto si no se tiene un buen método de exploración, la búsqueda se llega a atorar en un óptimo local.

Con base en los resultados obtenidos, el tiempo de ejecución de la implementación síncrona es considerablemente mayor que el tiempo de ejecución en la implementación asíncrono, pero la calidad de la solución es mucho mejor en la implementación síncrona. El uso de la implementación síncrona o asíncrona dependerá de las necesidades del usuario y del problema que esté atacando.

Para algunos problemas y para el modelo celular, se requirió del uso de una función que se encargara de mejor el proceso de exploración debido a la complejidad de los problemas o al comportamiento del modelo celular. Esta función es llama función de escape que se encargó de generar nuevos individuos en las poblaciones cuando ya no existía una mejora en la aptitud.

Sin las implementaciones paralelas de los modelos multipoblacionales, el tiempo de ejecución es considerable con forme se aumenta el número de poblaciones en el modelo distribuido y el tamaño de la malla en el modelo celular.

A pesar de que los valores de la eficiencia son bajos en algunos casos, la métrica de fracción serial nos dice que se tiene una buena paralelización debido a que este valor permanece constante según su definición.

Como experiencia propia, el desarrollo de esta biblioteca permitió realizar los experimentos de una forma sencilla, organizada y con mayor claridad de los resultados obtenidos.

Una vez que se tienen las funciones en este caso del algoritmo genético, ya no se requiere involucrarse en el código otra vez, simplemente variamos los parámetros en la interfaz gráfica y en ella misma se ejecutan las pruebas y se analizan los datos obtenidos en cualquier momento que se requiera.

Aportaciones

- Una propuesta de paralelización para el modelo distribuido del algoritmo genético y una propuesta de paralelización para el modelo celular del algoritmo genético.
- Una propuesta de paralelización para el algoritmo genético celular síncrono.

  Como se mencionó con anterioridad existen muy pocas bibliotecas que implementen modelos celulares, y las pocas que existen solamente manejan el modelo celular asíncrono, esto debido a la dificultad que conlleva la implementación síncrona.

- Una biblioteca para el modelado de problemas de metaheurísticas multipoblacionales en paralelo.

  Existen un gran número de implementaciones y bibliotecas que utilizan metaheurísticas, pero ese número se reduce considerablemente si se habla de metaheurísticas en el modelo distribuido y aún más si se trata de modelos celulares.

Trabajo futuro

- Mejorar las implementaciones utilizando estructuras de datos más eficientes.
- Implementar otras metaheurísticas como PSO, Hormigas, Abejas, entre otras.
- Agregar otras herramientas de análisis de los resultados.
- Experimentar con problemas multiobjetivo.
- Implementar modelos híbridos.
- Uso de diferentes metaheurísticas en diferentes poblaciones.

APÉNDICE

A.1 Tablas de resultados de experimentos

Experimento 1. Función Ackley

Escenario 1. AGD Varias poblaciones – Secuencial

Poblaciones	Tiempo	Generaciones	% Éxito	Promedio	Mejor
1	7.28	410	50%	0.0009	0.00025
5	1.76	109	63%	0.0008	0.000053
8	2.71	102	70%	0.0007	0.000107
10	3.37	100	90%	0.0005	0.000128

Tabla 9 - Tabla de resultados del experimento 1 - escenario 1

Escenario 2. AGD Varias poblaciones – Paralelo

Poblaciones	Tiempo	Generaciones	% Éxito	Promedio	Mejor
1	3.89	357	40%	0.0011	0.00020
5	0.57	97	80%	0.0007	0.000065
8	0.8	97	83%	0.0007	0.000117
10	1.01	96	90%	0.0005	0.000113

Tabla 10 - Tabla de resultados del experimento 1 - escenario 2

No. Islas	Aceleración	Eficiencia	Fracción Serial
1	1.87	0.93	0.0695187165
5	3.08	0.51	0.1896103896
8	3.38	0.37	0.2078402367
10	3.33	0.30	0.2303303303

Tabla 11 - Tabla de análisis de rendimiento paralelo del experimento 1 – escenarios 1 y 2

Escenario 3. AGC Variación del radio del vecindario – Secuencial

Radio	Tiempo	Generaciones	% Éxito	Promedio	Mejor
1	330.63	72	100%	0.0002248	0.00004957
3	544.73	116	96%	0.0004916	0.00003029
6	589.65	119	86%	0.0005519	0.00000774
10	515.91	104	86%	0.0005819	0.00001172
13	434.54	89	90%	0.0005599	0.00017213
15	448.62	92	90%	0.0005508	0.0001114

Tabla 12 - Tabla de resultados del experimento 1 - escenario 3

Escenario 4. AGC Variación del radio del vecindario – Paralelo

Radio	Tiempo	Generaciones	% Éxito	Promedio	Mejor
1	182.49	78	100%	0.0002878	0.0000121
3	191.42	79	93%	0.0004066	0.0000922
6	233.83	99	76%	0.000567	0.0000339
10	213.77	85	93%	0.0003976	0.0000058
13	209.3	78	90%	0.0004798	0.0000104
15	230.83	97	90%	0.0004622	0.0000514

Tabla 13 - Tabla de resultados del experimento 1 - escenario 4

Radio	Aceleración	Eficiencia	Fracción Serial
1	1.81	0.23	0.48793861
3	2.85	0.36	0.2587468
6	2.52	0.32	0.31035118
10	2.41	0.30	0.33069168
13	2.08	0.26	0.40761002
15	1.94	0.24	0.4451811

Tabla 14 - Tabla de análisis de rendimiento paralelo del experimento 1 – escenarios 3 y 4

Experimento 2. Problema del agente viajero (TSP)

Escenario 1. AGD Varias poblaciones – Secuencial

Poblaciones	Tiempo	Generaciones	% Éxito	Promedio	Mejor
1	549.37	5322	40%	28432.37	27601.17
5	443.23	1486	90%	27726.04	27601.17
8	639.81	1404	100%	27611.00	27601.17
10	769.75	1370	100%	27606.09	27601.17

Tabla 15 - Tabla de resultados del experimento 2 - escenario 1

Escenario 2. AGD Varias poblaciones – Paralelo

Poblaciones	Tiempo	Generaciones	% Éxito	Promedio	Mejor
1	275.64	5501	40%	28335.82	27601.17
5	121.21	1460	96%	27637.75	27601.17
8	134.81	1318	100%	27606.09	27601.17
10	173.25	1394	100%	27601.17	27601.17

Tabla 16 - Tabla de resultados del experimento 2 - escenario 2

No. Islas	Aceleración	Eficiencia	Fracción Serial
1	1.993070672	0.996535336	0.00347671
5	3.656711492	0.609451915	0.128163707
8	4.746012907	0.527334767	0.112041075
10	4.443001443	0.403909222	0.147580383

Tabla 17 - Tabla de análisis de rendimiento paralelo del experimento 2 – escenarios 1 y 2

Experimento 3. Problema de la mochila

Escenario 1. AGC Variación del número de procesos – Síncrono

Procesos	Tiempo	Generaciones	% Éxito	Promedio	Mejor
1	2657.51	1419	93%	5380	5623
4	696.76	1301	76%	5246.76	5623
6	599.76	1301	83%	5226.6	5623
8	619.22	1417	90%	5279.36	5623

Tabla 18 - Tabla de resultados del experimento 3 - escenario 1

Procesos	Aceleración	Eficiencia	Fracción Serial
1	1.00	1.00	1
4	3.81	0.95	0.156783176
6	4.43	0.74	0.115068509
8	4.29	0.54	0.123437245

Tabla 19 - Tabla de análisis de rendimiento paralelo del experimento 3 – escenario 1

Escenario 2. AGC Variación del número de procesos – Asíncrono

Procesos	Tiempo	Generaciones	% Éxito	Promedio	Mejor
1	2732.35	1476	83%	5323.2	5623
4	698.03	1279	80%	5188.16	5623
6	638.52	1355	90%	5250.33	5623
8	624.75	1427	86%	5284.96	5623

Tabla 20 - Tabla de resultados del experimento 3 - escenario 2

Procesos	Aceleración	Eficiencia	Fracción Serial
1	1.00	1.00	1
4	3.91	0.98	0.149107127
6	4.28	0.71	0.124215942
8	4.37	0.55	0.118456378

Tabla 21 - Tabla de análisis de rendimiento paralelo del experimento 3 – escenario 2

A.2 Configuración de MPHStudio

Para la correcta configuración de la biblioteca MPHStudio se requieren algunos valores de configuración. Estos valores de configuración nos permiten ejecutar los algoritmos genéticos, establecer la cantidad de procesos que utilizará el algoritmo genético celular así como los problemas que se desean trabajar con los algoritmos.

Los datos de configuración utilizan una base de datos relacional embebida SQLite. Este gestor de base de datos está contenido en una pequeña biblioteca escrita en C (Aproximadamente 275 KB), es por ello que se eligió este gestor.

Para establecer los parámetros de configuración, nos tenemos que dirigir al menú *Herramientas/Configuración*. La venta que se abre es como la que se muestra a continuación.

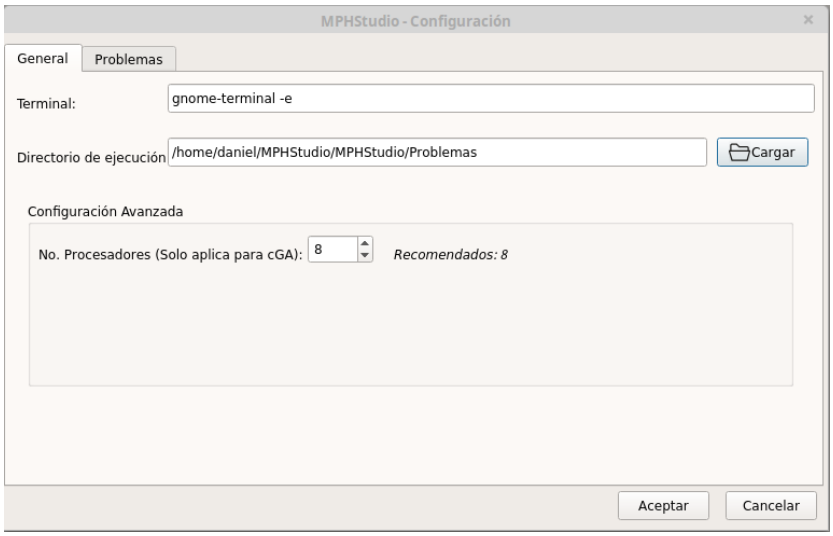


Figura 48 - Ventana de configuración general

La primera pestañan que se muestra es la pestaña de *General*, en esta pestaña se establecen parámetros básicos para el correcto funcionamiento de la biblioteca. La biblioteca requiere el nombre de la terminal del sistema, ya que para poder ejecutar los algoritmos genéticos, primero se crea un archivo temporal y luego se ejecuta en la terminal del sistema.

El directorio de ejecución, es el directorio donde se guardan los archivos de los problemas que se vayan registrando.

El número de procesadores es la cantidad de procesos que se crearan cuando se ejecute el algoritmo genético celular. La biblioteca muestra la cantidad de procesos recomendados basado en la cantidad de núcleos con los que cuenta el microprocesador. Es posible aumentar o reducir la cantidad de procesos; La mejora en rendimiento dependerá del problema, los datos con los que estemos trabajando y la cantidad de procesos que se estén ejecutando en el sistema operativo.

La siguiente pestaña es la pestaña de Problemas. En esta pestaña se agregan y quitan los problemas con los que se va a trabajar en la biblioteca MPHStudio.

En la siguiente imagen se muestra la ventana.

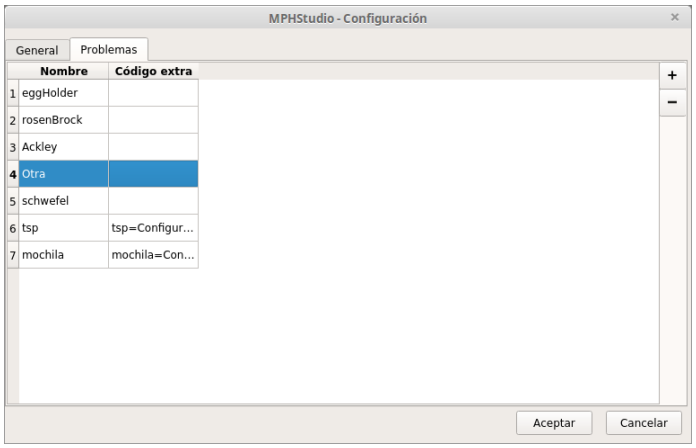


Figura 49 - Ventana de configuración de problemas

Al agregar un problema, automáticamente se cargarán en las ventanas de ejecución de los algoritmos.

A.3 Agregar un nuevo problema

Además de los problemas para prueba del sistema que se tienen implementados, la biblioteca MPHStudio tiene la capacidad de poder agregar otro problema y poder trabajarlo con algoritmo genético ya sea distribuido o celular.

Lo primero que se tiene que tener en cuenta es el tipo de problema con el que se está trabajando, la forma en que se representan las soluciones del problema y como se evalúan esas soluciones.

La biblioteca MPHStudio, proporciona funciones para problemas de optimización numérica y optimización combinatoria. Si estas funciones no les son útiles al investigador y/o desarrollador, se tendrán que implementar estas funciones adaptadas al tipo de problema con el que se está trabajando.

Las funciones que se requieren son las siguientes:

- Generación de la población inicial.
- Cruza de padres.
- Mutación de hijos.
- Muestra de resultados.

Las funciones a crear utilizando el lenguaje de programación Python, deben de importar las siguientes bibliotecas:

```
from MPHStudio.Objetos.Individuo import Individuo
from MPHStudio.Objetos.Poblacion import Poblacion
```

Las funciones deben ser métodos de una clase, por ejemplo class *ConfiguracionTSP*.

Función Generar Población

Esta función recibe como parámetros de entrada la configuración general, la configuración del problema y la cantidad total de individuos a crear.

EL valor que tiene que regresar es un arreglo de objetos del tipo individuo pertenecientes a la clase población.

Un ejemplo de esto es:

Función para generar la población inicial

```
def generaPoblacion(self,configuracion, confProblema,total):
    poblacion = Poblacion()
    for i in range(0,total):
        aux = Individuo(i)
        lista=range(len(confProblema.otrosDatos.aristas))
        np.random.shuffle(lista)

        aux.nDimensiones = len(confProblema.otrosDatos.aristas)
        aux.valores = lista
        aux.aptitud = confProblema.otrosDatos.calculaFitness(aux)
        poblacion.individuos.append(aux)

    return poblacion.individuos
```

Algoritmo 3 - Función de ejemplo para generar la población inicial

Función Cruza

La función de cruza recibe como parámetros dos objetos del tipo individuos que son los valores de cada uno de los padres a ser cruzados.

Los valores que debe de regresar son dos objetos del tipo individuos que son los hijos generados a partir de la cruza de los padres.

Como ejemplo de esta función tenemos la cruza ordenada de un problema del tipo combinatorio.

Función cruza

```
def cruzaOrdenada(self,padre1, padre2):
    #Copio los valores
    hijo1 = Individuo(-1)
    hijo2 = Individuo(-1)
    hijo1.nDimensiones = padre1.nDimensiones
    hijo2.nDimensiones = padre2.nDimensiones

    # Cruza Ordenada
    punto1 = np.random.randint(padre1.nDimensiones)
    punto2 = punto1
    while punto2 == punto1:
        punto2 = np.random.randint(padre1.nDimensiones)

    if punto2<punto1:
        aux=punto1
        punto1 = punto2
        punto2 = aux

    centro1 = padre1.valores[punto1:punto2]
    centro2 = padre2.valores[punto1:punto2]
    # Ordeno los centros
    auxP1 = []
    auxP2 = []
    for i,j in zip(padre1.valores,padre2.valores):
        if i in centro2:
            auxP2.append(i)
        if j in centro1:
```

```
            auxP1.append(j)
    centro1 = auxP1[:]
    centro2 = auxP2[:]

    #Creo los hijos
    hijo1.valores = padre1.valores[:punto1]+auxP1+padre1.valores[punto2:]
    hijo2.valores = padre2.valores[:punto1]+auxP2+padre2.valores[punto2:]

    return  hijo1, hijo2
```

Algoritmo 4 - Función de ejemplo para cruzar dos individuos

Función mutación

La función de mutación recibe como parámetros un objeto del tipo individuo y la configuración delk problema.

El valor que regresa es un objeto del tipo individuo.

Como ejemplo tenemos la función de mutación por inserción.

Función mutación

```
def mutacionInsercion(self,hijo1,confproblema):
    #Mutación por insert
    for i in range(0,self.nVecesMuta):
        aleatorio=np.random.random()
        pos1=np.random.randint(0,math.ceil(hijo1.nDimensiones/2))
        while(aleatorio<=self.porcentajeMutacion):
            aleatorio=np.random.random()
            pos1=np.random.randint(0,math.ceil(hijo1.nDimensiones/2))

        aleatorio=np.random.random()
        pos2=np.random.randint(math.ceil(hijo1.nDimensiones/2),hijo1.nDimensiones)
        while(aleatorio<=self.porcentajeMutacion):
            aleatorio=np.random.random()

            pos2=np.random.randint(math.ceil(hijo1.nDimensiones/2),
                 hijo1.nDimensiones)
        #Insert
        tmp=hijo1.valores[pos2]
        del[hijo1.valores[pos2]]
        hijo1.valores.insert(pos1+1,tmp)

    hijo1.aptitud=confproblema.otrosDatos.calculaFitness(hijo1)
    return hijo1
```

Algoritmo 5 - Función de ejemplo para mutar un individuo

Función mostrar resultados

En algunos problemas se requiere que se muestre los resultados de una forma específica, para ello se puede implementar una función que lo haga.

El valor que recibe como parámetro es un objeto del tipo Individuo.

En el siguiente ejemplo se muestra una forma de ser implementada.

Función mostrar resultados

```
def imprimeResultado(self,Individuo):
    corte=Individuo.longitud
    print(Individuo.valores[:corte])
    sumatoriaPeso=0
    sumatoriaValor=0
    puntoCorte=0
    for i in Individuo.valores:
        valor=i
        if self.tablaDatos[valor-1][1]+sumatoriaPeso>self.pesoMaximo:
            break
        sumatoriaPeso+=self.tablaDatos[valor-1][1]
        sumatoriaValor+=self.tablaDatos[valor-1][0]
        puntoCorte+=1
    print ("Peso: "+str(sumatoriaPeso))
```

Algoritmo 6 - Función de ejemplo para mostrar resultados de una forma específica

Una vez implementadas estas funciones, se deben de guardar en un archivo único y registrarse en la configuración de MPHStudio.

Para establecer los parámetros de configuración, nos tenemos que dirigir al menú *Herramientas/Configuración*. La venta que se abre es como la que se muestra a continuación.

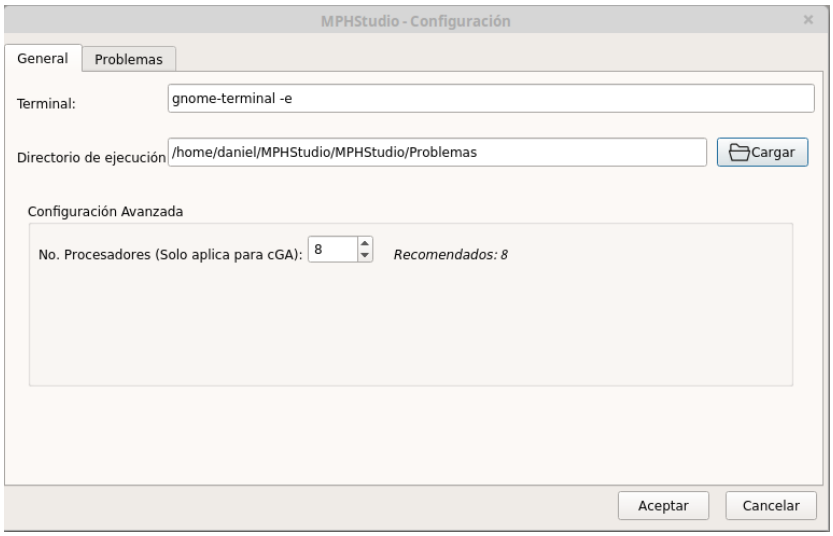


Figura 50 - Ventana de configuración MPHStudio

La primera pestañan que se muestra es la pestaña de *General*

La siguiente pestaña es la pestaña de Problemas. En esta pestaña se agregan y quitan los problemas con los que se va a trabajar en la biblioteca MPHStudio.

En la siguiente imagen se muestra la ventana.

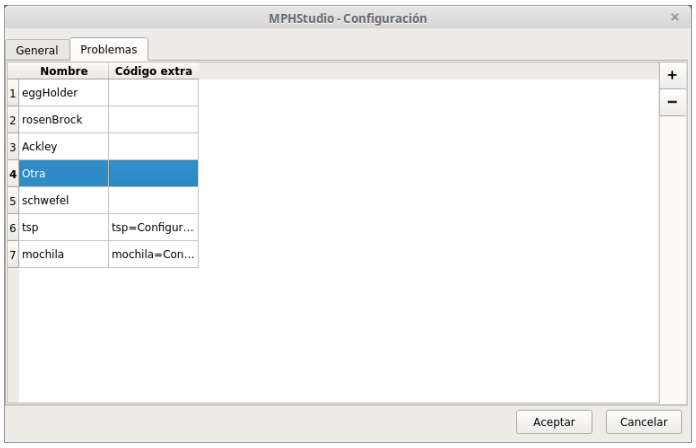


Figura 51 - Ventana de configuración de MPHStudio - Problemas

Al agregar un problema, automáticamente se cargarán en las ventanas de ejecución de los algoritmos.

Para agregar un problema damos clic en el botón +. Al hacer esto se abrirá una ventana donde se requerirán algunos datos.

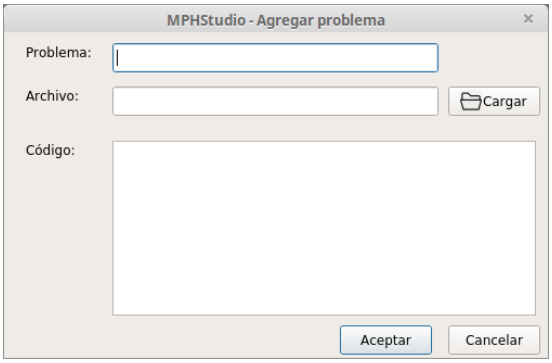


Figura 52 - Ventana para agregar problema de MPHStudio

En el cuadro de problema ahí debemos poner el nombre del problema con el que será identificado. El *Archivo* es un archivo que se adjunta el cual contiene implementaciones propias de las diferentes funciones del algoritmo genético así como el modo de representación de los individuos. Este archivo se importa al inicio del código que se genera a la hora de ejecutar alguna de las implementaciones de

algoritmos genéticos propuestas. El código son líneas adicionales que se añaden al final del código generado por la herramienta.

Un ejemplo de este código es como el que se muestra a continuación.

Código adicional de ejemplo del problema de la mochila en python
mochila=ConfiguracionMochila() mochila.cargarMochila("MochilaData/mochila3.moc") mochila.nVecesMuta=2 mochila.nPosiciones=1 mochila.porcentajeMutacion=0.7

Algoritmo 7 - Código adicional de ejemplo del problema de la mochila en python

Una vez que se agregó el problema deseado, en cualquiera de las implementaciones de los algoritmos genéticos se cargaran automáticamente como se mencionó con anterioridad.

Siguiendo el ejemplo anterior, vemos que existe una variable llamada *mochila*, esa variable habrá que establecerla a la hora de correr los algoritmos. Además de ese dato, debemos establecer el nombre de las funciones que agregamos; esto principalmente aplica para problemas de optimización combinatoria. La imagen siguiente muestra un ejemplo.

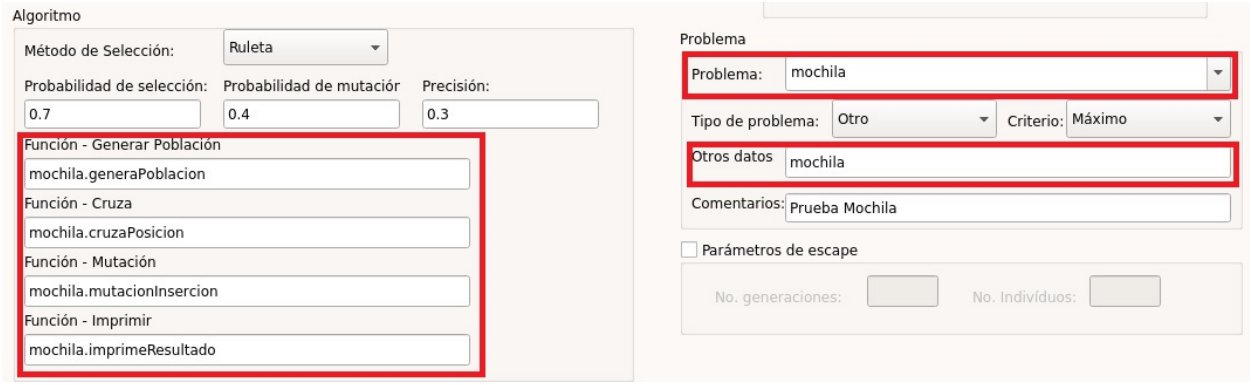


Figura 53 - Selección del problema a trabajar y los datos a llenar

A.4 Manual de usuario

La biblioteca MPHStudio cuenta con diferentes módulos que permiten ejecutar el algoritmo genético distribuido y el algoritmo genético celular en paralelo así como un conjunto de herramientas que permiten analizar el comportamiento y los resultados de ejecución de estos algoritmos.

Para poder hacer uso de esta biblioteca se presenta este manual con la explicación rápida y concisa de la forma de ejecutar los algoritmos y la mejor manera de utilizar las herramientas con las que cuenta esta biblioteca, así como una explicación de cómo agregar más herramientas y más algoritmos a esta biblioteca.

Ejecución del algoritmo genético distribuido

Para realizar una ejecución del algoritmo genético distribuido, debemos dar clic en el botón *Ejecutar Isla* del menú de herramientas de la pantalla principal.

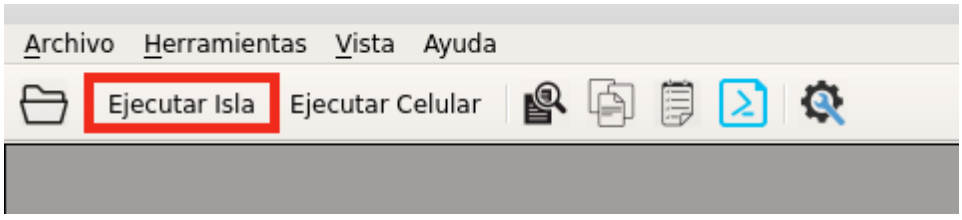


Figura 54 - Botón Ejecutar Isla de la barra de herramientas

Al presionar el botón se abrirá la ventana de ejecución de islas como la que se muestra en la siguiente figura. En esta ventana tenemos que especificar cada uno de los parámetros que requiere la sección de *Islas, Configuración de la población, Algoritmo y problema.*

Para la sección de *Algoritmo*, si se van a probar funciones de optimización numérica, solamente es necesario seleccionar el método de selección, la probabilidad de selección, la probabilidad de mutación y la precisión. Si no se tiene claro los porcentajes a utilizar, se puede iniciar con una probabilidad de 0.5 y comprobar si ayuda a la obtención del óptimo global. En esta misma sección si se prueban funciones de optimización combinatoria, es necesario especificar cada uno de las funciones que se utilizarán para la ejecución del algoritmo.

La biblioteca MPHStudio, cuenta con algunos problemas de ejemplo tanto de optimización numérica como de optimización combinatoria, así como algunos archivos de ejemplo de configuración con los cuales es posible comprender mejor su funcionamiento.

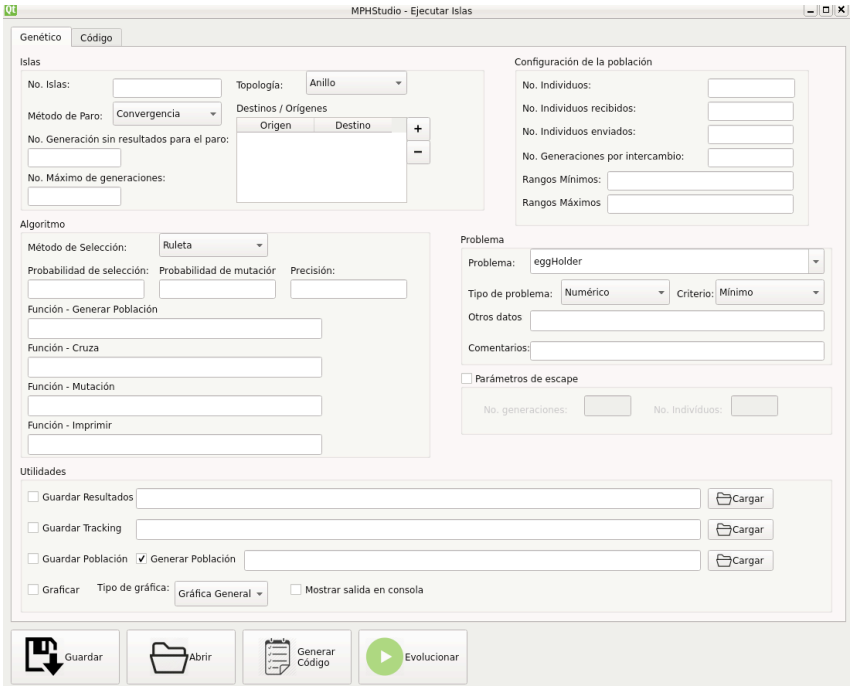


Figura 55 - Ventana de ejecución de modelo distribuido

Con esta interfaz se puede hacer uso de las diferentes utilidades que se tienen a la hora de ejecutar cada uno de los algoritmos, estas utilidades nos permiten guardar los resultados finales de ejecución, guardar el archivo de tracking que nos permite analizar el comportamiento que tuvo la ejecución del algoritmo, guardar la población inicial así como cargar una población iniciar previamente generada, mostrar la gráfica de convergencia al final de la ejecución y mostrar los resultados en consola.

Si se requiere guardar los parámetros establecidos, se debe presionar el botón guardar, de igual forma para abrir algún archivo de parámetros basta con presionar el botón abrir que se encuentra en la parte inferior de esta ventana.

En esta ventana es posible generar código Python para su implementación en proyectos propios utilizando los parámetros establecidos en esta interfaz; basta con presionar el botón *Generar Código,* y se mostrará el código generado para ser copiado.

Una vez explicado estas funcionalidades, basta con presionar el botón Evolucionar para ejecutar el algoritmo genético distribuido con los parámetros establecidos y una ventana de consola se abrirá mostrando el inicio y fin de la ejecución.

Ejecución del algoritmo genético celular

Para realizar una ejecución del algoritmo genético celular, debemos dar clic en el botón *Ejecutar Celular* del menú de herramientas de la pantalla principal.

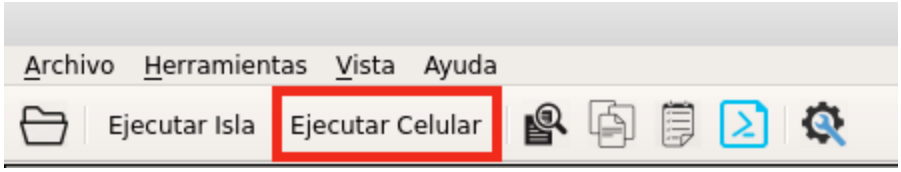


Figura 56 - Botón Ejecutar Celular de la barra de herramientas

Al presionar el botón se abrirá la ventana de ejecución del modelo celular como la que se muestra en la siguiente figura. En esta ventana tenemos que especificar cada uno de los parámetros que requiere la sección de *Celular, Configuración de la población, Algoritmo y problema.*

Así como para el modelo distribuido, para la sección de *Algoritmo*, si se van a probar funciones de optimización numérica, solamente es necesario seleccionar el método de selección, la probabilidad de selección, la probabilidad de mutación y la precisión. Si no se tiene claro los porcentajes a utilizar, se puede iniciar con una probabilidad de 0.5 y comprobar si ayuda a la obtención del óptimo global. En esta misma sección si se prueban funciones de optimización combinatoria, es necesario especificar cada uno de las funciones que se utilizarán para la ejecución del algoritmo.

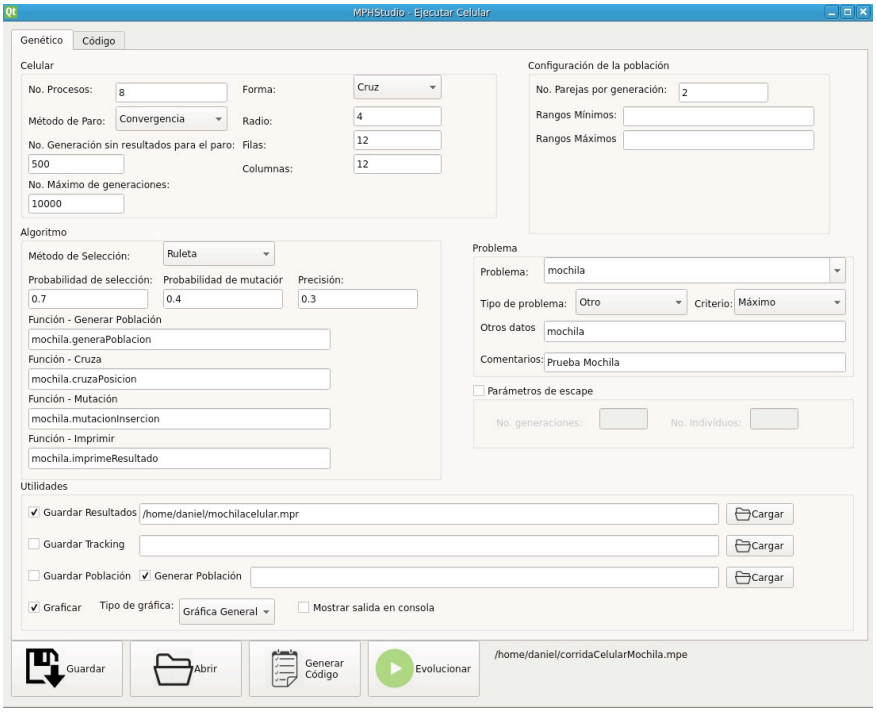


Figura 57 - Ventana de ejecución de modelo celular

Con esta interfaz, al igual que en el modelo distribuido, se puede hacer uso de las diferentes utilidades que se tienen a la hora de ejecutar cada uno de los algoritmos, estas utilidades nos permiten guardar los resultados finales de ejecución, guardar el archivo de tracking que nos permite analizar el comportamiento que tuvo la ejecución del algoritmo, guardar la población inicial así como cargar una población iniciar previamente generada, mostrar la gráfica de convergencia al final de la ejecución y mostrar los resultados en consola.

Si se requiere guardar los parámetros establecidos, se debe presionar el botón guardar, de igual forma para abrir algún archivo de parámetros basta con presionar el botón abrir que se encuentra en la parte inferior de esta ventana.

En esta ventana es posible generar código Python para su implementación en proyectos propios utilizando los parámetros establecidos en esta interfaz; basta con presionar el botón *Generar Código,* y se mostrará el código generado para ser copiado.

Presione el botón *Evolucionar* para ejecutar el algoritmo genético distribuido con los parámetros establecidos y una ventana de consola se abrirá mostrando el inicio y fin de la ejecución.

Hay que tener en cuenta que al tener un tamaño muy grande de malla, el tiempo de ejecución del algoritmo será considerable y la eficiencia que pueda tener al ejecutar este algoritmo va a depender de la cantidad de procesadores con los que se cuenta a la hora de la ejecución.

Herramientas de MPHStudio

Herramienta de análisis de resultados

Para ejecutar la herramienta de análisis de datos damos clic en el botón  de la barra de herramientas de la pantalla principal.

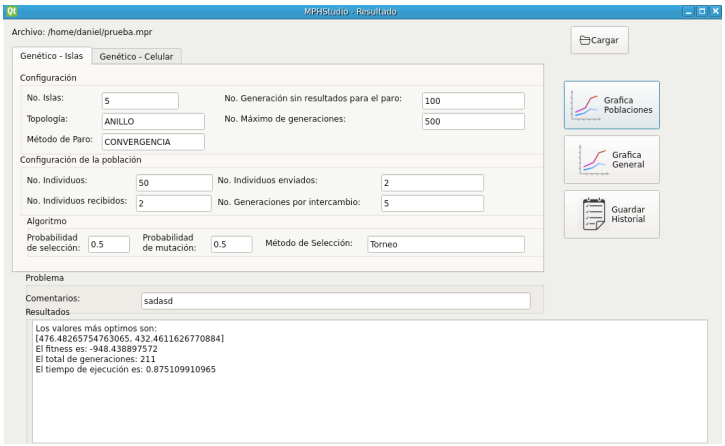


Figura 58 - Herramienta de análisis de resultados

Al abrirse la ventana de análisis de resultados, damos clic en el botón *Cargar* para abrir un archivo de resultados. Una vez que se haya abierto el archivo, los parámetros con que se ejecutó el algoritmo se mostraran en esta ventana así como los resultados finales de la ejecución.

Con el botón *Gráfica Poblaciones* podemos obtener la gráfica de convergencia seleccionando que poblaciones mostrar y si damos clic en el botón *Gráfica General* se mostrará la gráfica de convergencia promedio.

Por último el botón *Guardar Historial* permite guardar en un archivo CSV las mejor aptitudes de cada generación.

Herramienta de bitácora de comparación de resultados

Para abrir esta herramienta damos clic en el botón  de la barra de herramientas de la pantalla principal.

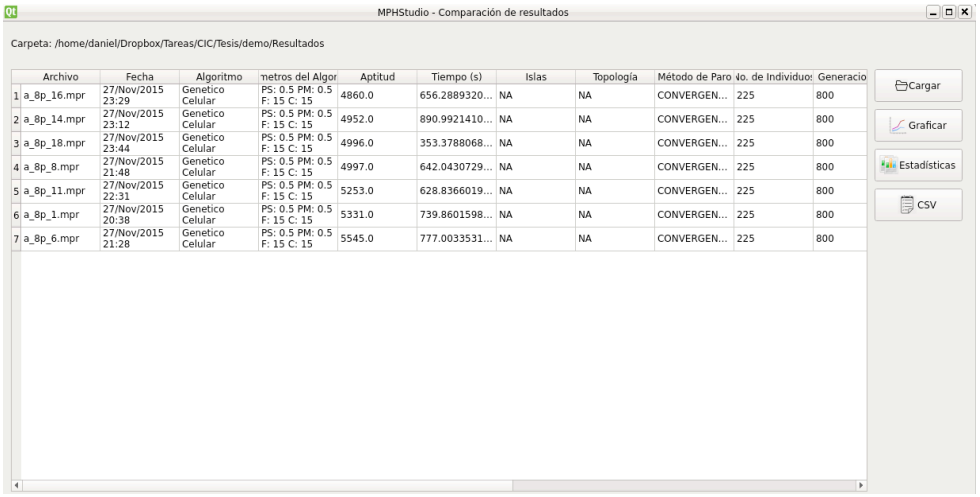


Figura 59 - Herramienta de comparación de resultados

Para hacer la comparación entre diferentes resultados debemos abrir la carpeta donde se encuentran los archivos de resultados. Para abrir la carpeta damos clic en el botón *Cargar* y resultados se cargarán en la tabla de esta ventana.

Para graficar seleccionamos los archivos que queremos graficar y damos clic en el botón *Graficar*. De igual manera si se quieren obtener estadísticas seleccione los archivos y de clic en el botón *Estadísticas*.

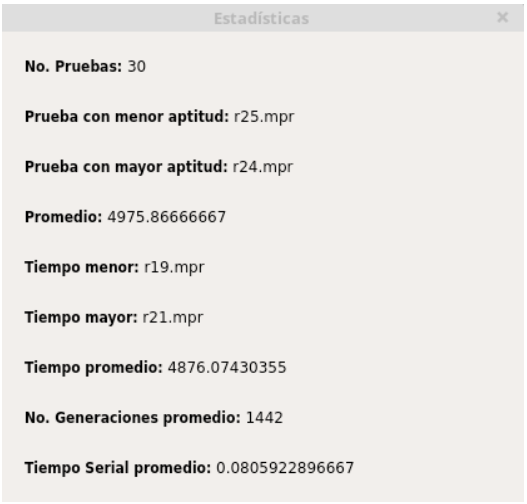


Figura 60 - Ventana de estadísticas

Una ventana se abrirá como la imagen anterior donde se muestran datos como el tiempo promedio, generaciones promedio, tiempo menor, tiempo mayor, etc.

Si la tabla de datos se quiere pasar a un archivo CSV, seleccionamos los archivos que queremos que se guarden y damos clic en el botón *CSV*.

Herramienta de análisis del comportamiento del algoritmo

Damos clic en el botón  para abrir la herramienta de análisis del comportamiento del algoritmo.

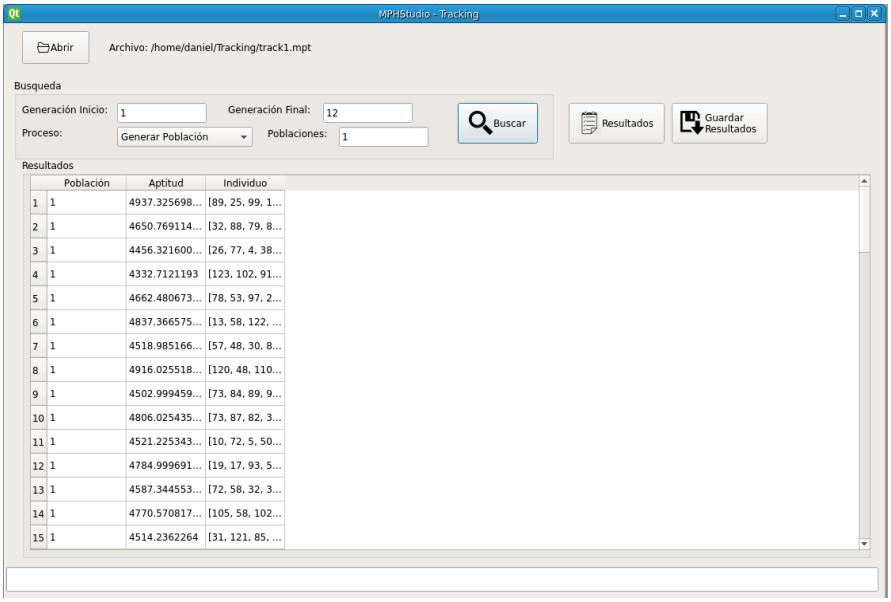


Figura 61 - Herramienta de análisis del comportamiento del algoritmo

Para cargar un archivo, damos clic en el botón *Abrir* y esperamos a que se carguen los datos. Una vez cargados los datos, podemos realizar una búsqueda estableciendo el rango de generaciones a buscar, el proceso y las poblaciones a observar. Una vez establecidos esos datos damos clic en el botón *Buscar* y los datos con los filtros establecidos se mostrarán en la tabla de resultados

Esta herramienta tiene la opción de mostrar los resultados finales, para ello solamente presione el botón *Resultados* y la ventana de análisis de resultados se mostrará.

Herramienta de múltiples ejecuciones

Para ejecutar esta herramienta damos clic en el botón  . Una ventana como la siguiente imagen se abrirá con la cual podemos seleccionar el archivo de parámetros con que se ejecutarán las corridas, el directorio donde se guardaran los archivos de resultados y la cantidad de ejecuciones a correr. En esta venta también se puede establecer si las corridas de los algoritmos serán en forma secuencial o de forma paralela.

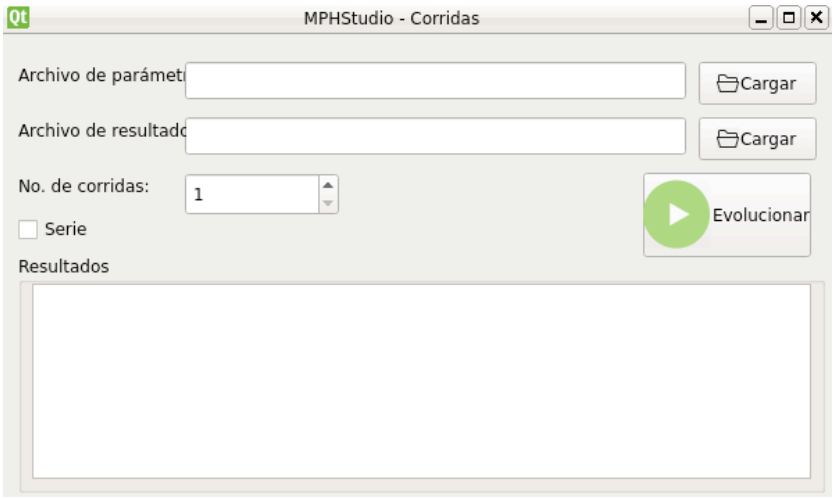


Figura 62 - Herramienta de múltiples ejecuciones

Ya que todos los parámetros se establecieron damos clic en el Botón *Evolucionar* para que comiencen las ejecuciones y los resultados se mostrarán en el cuadro inferior.

Agregar nueva metaheurística

Para poder implementar una nueva metaheurística se debe tener en cuenta los diferentes módulos con los que cuenta la biblioteca MPHStudio, ya que para hacer uso pleno de todas las herramientas se debe agregar ciertas funciones y hacer referencia a estas desde la interfaz principal de la biblioteca.

Para iniciar tenemos que seleccionar el modelo multipoblacional que se desea ocupar, ya sea el modelo distribuido o el modelo celular.

Necesitamos crear una clase que herede de la clase *MPHStudio.Multipoblaciones.Islas* para el caso del modelo distribuido o de la clase *MPHStudio.Multipoblaciones.Celular* para el caso de modelo celular.

Una vez creadas esas clases debemos implementar algunas funciones, principalmente la función *evolucionar()*, en la cual se realizará el proceso de evolución. Un ejemplo de esto es como se muestra a continuación.

Ejemplo para implementar una nueva metaheurística

```
class nuevaMetaheuristica(MPHStudio.Multipoblaciones.Islas):
   def __init__(self):
      MPHStudio.Multipoblaciones.Islas.__init__(self)
      MPHStudio.Multipoblaciones.Islas.metaheuristica=self

   def evolucionar(self):
     #Funciones evolutivas
     #Guardo el mejor individuo en el individuo 0 de la población
     #self.poblacion.individuos[0]
```

Algoritmo 8 - Clase de ejemplo para una nueva metaheurística

Para arrancar la ejecución debemos hacer una llamada a la función *nuevaMetaheuristica.inicioIslas(configuración,configuracionProblema),* como se muestra en el siguiente ejemplo.

Arranque de la nueva metaheurística

```
from mpi4py import MPI
import sys
import MPHStudio.Constantes
from MPHStudio.Objetos.Islas import ConfiguracionIslas
from MPHStudio.Objetos.Poblacion import ConfiguracionPoblacion
from MPHStudio.Metaheuristicas.nuevaMetaheuristica import nuevaMetaheuristica
from MPHStudio.Problemas.TSP import *
from MPHStudio.Problemas.Problemas import Problema
from MPHStudio.Util import cargarDatos
def main():
nPoblaciones = MPI.COMM_WORLD.Get_size()-1
#Configuración de las Islas
configIslas=ConfiguracionIslas()
configIslas.nIslas=nPoblaciones
```

```
configIslas.topologia= Constantes.TOPOLOGIA_ANILLO
configIslas.metodoParo= Constantes.METODO_PARO_CONVERGENCIA
#Número máximo de generaciones sin cambio en la aptitud del mejor individuo
configIslas.nGeneracionesNCParo=100
#Número máximo de generaciones por ejecución
configIslas.nGeneracionesMaximasParo=20000
#Utilidades
configIslas.guardarDatosTXT=False
configIslas.archivoDatos="resultado1.txt"
configIslas.guardarResultados=True
configIslas.archivoResultados="Resultados/resultado_PGA10_DEMO.mpr"
configIslas.debug=True
configIslas.grafica=True
configIslas.tipoGrafica= Constantes.GRAFICA_AVG


configIslas.guardaPoblacion=False
configIslas.generaPoblacion=True
configIslas.archivoPoblacion="poblacion1.pbl"
configIslas.guardarTracking=True
configIslas.archivoTracking="/home/daniel/Tracking/trackDEMO.mpt"
#Configuración Poblaciones
confPoblacion=ConfiguracionPoblacion()
confPoblacion.nIndividuos=50
confPoblacion.nIndividuosIntercambio=3
confPoblacion.nIndividuosRecibidos=3
confPoblacion.nGeneracionesIntercambio=10
#Le pongo la misma configuración a todas las poblaciones
configIslas.configuracionPoblaciones=[confPoblacion]*nPoblaciones

#Configuración del TSP
tsp=ConfiguracionTSP()#10 Ciudades
tsp.cargarTSPLib("TSPData/dj38.tsp")
#tsp.cargarTSPSolucion("st70.opt.tour")
tsp.nVecesMuta=1
tsp.nPosiciones=1
tsp.porcentajeMutacion=0.7

#Configuración del problema
confproblema=Problema()
confproblema.problema=eggHolder
confproblema.tipoProblema= Constantes.PROBLEMA_COMBINATORIO
confproblema.otrosDatos=tsp
confproblema.criterio= Constantes.MINIMO
confproblema.comentarios="Problema TSP xqf131.tsp -1 veces mutacion - ESCAPE 400-
100"

#Nueva Metaheurística
nuevaMetaheuristica=nuevaMetaheuristica()
nuevaMetaheuristica.inicioIslas(configIslas,confproblema)
```

Algoritmo 9 - Ejemplo de arranque de la nueva metaheurística

Para implementar la nueva metaheurística en la interfaz gráfica de MPHStudio debemos crear una ventana ya sea para el modelo distribuido o el modelo celular. Puede tomar como referencia los archivos *MPHStudio.Vistas.uiRunIslas.py* para el modelo distribuido y *MPHStudio.Vistas.uiRunCelular.py* para el modelo celular.

Estas ventanas deben ser agregadas a la función *agregarBotones* del archivo *MPHStudio.Vistas.uiPrincipal.py*

El código de ejemplo para agregar a la función *agregarBotones()* es como el siguiente:

Código para agregar un botón

```
#Boton nuevo
self.nuevoBoton = QtWidgets.QAction(MainWindow)
self. nuevoBoton.setObjectName("nuevoBoton")
self. nuevoBoton.triggered.connect(self.funcion_que_maneja_el_boton)
self. nuevoBoton.setToolTip("Nuevo Botón")
self. nuevoBoton.setIcon(icon5)
self.toolBar.addAction(self. nuevoBoton)
```

Algoritmo 10 - Código de ejemplo para agregar un botón a la interfaz principal

Cada una de las herramientas con las que cuenta MPHStudio fueron creadas para trabajar exclusivamente con el algoritmo genético, es por ello que para agregar una nueva metaheurística se tiene que hacer una copia de los archivos siguientes ajustando los parámetros dependiendo de la metaheurística:

- Herramienta de análisis de resultados - *MPHStudio.Vistas.uiResultado.py*
- Herramienta de bitácora de comparación de resultados - *MPHStudio.Vistas.uiResultados.py*
- Herramienta de análisis de comportamiento del algoritmo - *MPHStudio.Vistas.uiTracking.py*
- Herramienta de múltiples corridas - *MPHStudio.Vistas.uiCorridas.py*

Una vez realizadas las copias de esos archivos, podemos agregar las nuevas ventanas a la interfaz principal de la misma forma en que se agrega la interfaz de la nueva metaheurística.

Como comentario final, puede revisar la implementación del algoritmo genético distribuido en el archivo *MPHStudio.Metaheuristicas.GC.py* y la implementación del algoritmo genético celular en el archivo *MPHStudio.Metaheuristicas.GCCelular.py*.

Referencias

[1] K. Manda, S. C. Satapathy y B. Poornasatyanarayana, «Population based meta-heuristic techniques for solving optimization problems: A selective survey,» *International Journal of Emerging Technology and Advanced Engineering,* vol. 2, nº 11, pp. 206-211, 2012.

[2] E. Alba, G. Luque y S. Nesmachnow, «Parallel metaheuristics: recent advances and new trends,» *International Transactions in Operational Research,* vol. 20, nº 1, pp. 1-48, 2013.

[3] F. M. Márquez, UNIX: Programación Avanzada, España: Alfaomega - Ra-Ma, 2006.

[4] E. Alba y A. J. Nebro, «New Technologies in Parallelism,» de *Parallel Metaheuristics: A New Class of Algorithms*, New Jersey, John Wiley & Sons, Inc, 2005.

[5] OpenMP, «OpenMP: Application Program Interface,» Julio 2013. [En línea]. Available: http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf. [Último acceso: 15 Agosto 2015].

[6] Z. Long, L. Yanchao, G. Song y X. Cheng-Zhong, «Architecture-based design and optimization of genetic algorithms on multi- and many-core systems,» *Future Generation Computer Systems,* nº 38, pp. 74-91, 2014.

[7] G. Sena, G. Isern y D. Megherbi, «Implementation of a parallel Genetic Algorithm on a cluster of workstations: Traveling Salesman Problem, a case study,» *Future Generation Computer Systems,* vol. 17, pp. 477-488, 2001.

[8] M. Rucinski, D. Izzo y F. Biscani, «On the Impact of the Migration Topology on the Island Model,» *Parallel Computing,* vol. 36, pp. 555-571, 2010.

[9] I. Andalon-Garcia y A. Chavoya, «Performance comparison of three topologies of the island model of a parallel genetic algorithm implementation on a cluster platform,» de *Electrical Communications and Computers (CONIELECOMP), 2012 22nd International Conference on*, Puebla, 2012.

[10] R. Banos, C. Gil, B. Paechter y J. Ortega, «Parallelization of population-based multi-objective meta-heuristics: An empirical study,» *Applied Mathematical Modelling,* vol. 30, nº 7, pp. 578--592, 2006.

[11] E. Alba y J. M. Troya, «Analyzing synchronous and asynchronous parallel distributed genetic algorithms,» *Future Generation Computer Systems,* vol. 17, nº 4, pp. 451 - 465, 2001.

[12] E. Alba, B. Dorronsoro, M. Giacobini y M. Tomassini, «Decentralized cellular evolutionary algorithms,» *Handbook of Bioinspired Algorithms and Applications,* vol. 7, pp. 103-120, 2005.

[13] F. Pinel, B. Dorronsoro y P. Bouvry, «A new parallel asynchronous cellular genetic algorithm for scheduling in grids,» de *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, IEEE International Symposium on, 2010, pp. 1-8.

[14] B. Andrés y Toro, «El aprendizaje con algoritmos genéticos,» de *Aprendizaje Automático: Un enfoque práctico*, España, Ra-Ma, 2010, pp. 263-292.

[15] J. Pinho, J. L. Sobral y M. Rocha, «Parallel evolutionary computation in bioinformatics applications,» *Computer methods and programs in biomedicine,* vol. 110, nº 2, pp. 183-191, 2013.

[16] The MathWorks, Inc, «Global Optimization Toolbox - Mathworks,» The MathWorks, Inc, [En línea]. Available: http://www.mathworks.com/products/global-optimization/. [Último acceso: 20 10 2015].

[17] E. Alba, «Mallba Library v2.0,» [En línea]. Available: http://neo.lcc.uma.es/mallba/easy-mallba/index.html. [Último acceso: 25 7 2014].

[18] A. Lee, «inspyred: Bio-inspired Algorithms in Python,» University of Arkansas, [En línea]. Available: http://pythonhosted.org/inspyred/index.html. [Último acceso: 01 05 2015].

[19] D. Dyeer, «Watchmaker Framework for Evolutionary Computation,» [En línea]. Available: http://watchmaker.uncommons.org/. [Último acceso: 8 2014].

[20] D. Izzo y F. Biscani, «The Python Parallel Global Multiobjective Optimizer,» [En línea]. Available: http://esa.github.io/pygmo/. [Último acceso: 11 2014].

[21] M. R. Garey y D. S. Johnson, Computers and intractability: a guide to the theory of NP-completeness, W.H. Freeman and company, 1979.

[22] C. Blum, A. Roli y E. Alba, «An introduction to metaheuristic techniques,» de *Parallel metaheuristics: A new class of algorithms*, New Jersey, John Wiley & Sons, Inc, 2005.

[23] T. Stützle, Local Search Algorithms for Combinatorial Problems - Analysis, Algorithms and New Applications, Germany: Infix Sankt Augustin, 1999.

[24] R. Tanese, «Distributed genetic algorithms,» de *Proceedings of the Thrid International Conference on Genetic Algorithms*, 1989.

[25] B. Manderick y P. Spiessens, «Fine-grained parallel genetic algorithm,» de *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[26] M. Flynn, «Very high-speed computing systems,» *Proceedings of the IEEE,* vol. 54, nº 12, pp. 1901-1909, 1966.

[27] A. S. Tanenbaum, Distributed Operating Systems, Prentice-Hall, 1995.

[28] A. G. R, Foundations of Multithreaded, Parallel and Distributed Programming, Addison Wesley, 2000.

[29] M. J. Quinn, Parallel Programming in C with MPI y OpenMP, McGraw-Hill, 2003.

[30] E. Alba y G. Luque, «Measuring the Performance of Parallel Metaheuristics,» de *Parallel metaheuristics: A new class of algorithms*, New Jersey, John Wiley & Sons, Inc, 2005, pp. 43-92.

[31] G. M. Amdahl, «Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities,» *Solid-State Circuits Society Newsletter, IEEE,* vol. 12, nº 3, pp. 19-20, 2007.

[32] G. E. Karniadakis y R. M. Kirby, Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation, Cambridge University Press, 2003.

[33] A. H. Karp y H. P. Flatt, «Measuring Parallel Processor Performance,» *Commun. ACM,* vol. 33, nº 5, pp. 539--543, 1990.

[34] A. López, «Diseño de un algoritmo evolutivo multiobjetivo paralelo,» IPN, México, 2005.

[35] E. Alba, E.-G. Talbi, G. Luque y N. Melab, «Metaheuristics and Parallelism,» de *Parallel metaheuristics: A new class of algorithms*, New Jersey, John Wiley & Sons, Inc, 2015.

[36] J. Brownlee, Clever Algorithms, Melbourne, Australia: Jason Brownlee, 2011.

[37] E. Alba y M. Tomassini, «Parallelism and evolutionary algorithm,» *Evolutionary Computation, IEEE Transactions on,* vol. 6, nº 5, pp. 443--462, 2002.

[38] T. Back, D. B. Fogel y Z. Michalewicz, Handbook of evolutionary computation, IOP Publishing Ltd., 1997.

[39] D. E. Goldberg y K. Deb, «A comparative analysis of selection schemes used in genetic algorithms,» *Foundations of genetic algorithms,* vol. 1, pp. 69-93, 1991.

[40] E. Alba y J. M. Troya, «A survey of parallel distributed genetic algorithms,» *Complexity,* vol. 4, nº 4, pp. 31-52, 1999.

[41] G. Luque, E. Alba y B. Dorronsoro, «Parallel Genetic Algorithms,» de *Parallel metaheuristics: A new class of algorithms*, Ney Jersey, John Wiley & Sons, Inc.

[42] E. Alba y B. Dorronsoro, Cellular genetic algorithms, Springer Science & Business Media, 2009.

[43] D. Baezley, «PyCon,» de *Understanding the Python GIL*, Atlanta. EUA, 2010.

[44] Python Software Foundation, «Python 2.7.10 documentation,» 07 Septiembre 2015. [En línea]. Available: https://docs.python.org/2/index.html. [Último acceso: 14 Diciembre 2014].

[45] J. Liang, B. Qu, P. Suganthan y A. G. Hernández-Díaz, «Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization,» *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report,* vol. 201212, 2013.

[46] R. Johnsonbaugh, «Ciclos hamiltonianos y el problema del agente viajero,» de *Matemáticas Discretas*, Pearson Educación, 2005, pp. 340-345.

[47] R. M. Karp, Reducibility among combinatorial problems, Springer, 1972.

[48] C. H. Papadimitriou y K. Steiglitz, Combinatorial Optimization - Algorithms and Complexity, New York: Dover Publications, Inc, 1982.