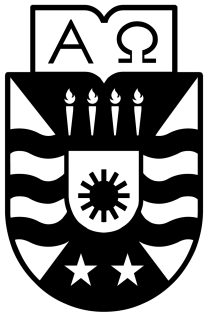


UNIVERSIDAD DEL BÍO-BÍO

FACULTAD DE INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE UN ALGORITMO GENÉTICO PARALELO PARA LA SOLUCIÓN DE PROBLEMAS DE OPTIMIZACIÓN COMBINATORIA

TRABAJO DE TÍTULO QUE PRESENTA:

JOSÉ NICOLÁS FONSECA AVENDAÑO

PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN AUTOMATIZACIÓN

DIRECTORES: CRISTIAN DURÁN FAÚNDEZ
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
UNIVERSIDAD DEL BÍO-BÍO

CARLOS HERRERA LÓPEZ
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL
UNIVERSIDAD DE CONCEPCIÓN

VÍCTOR PARADA DAZA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

CONCEPCIÓN-CHILE SEPTIEMBRE DE 2015

AGRADECIMIENTOS

En primer lugar quisiera agradecer a mi familia quienes siempre me han apoyado independiente de los problemas. En especial a mis padres, mi madre que siempre estuvo preocupada por mí, evitando presionarme y mostrando interés por este trabajo, a mi padre por el apoyo económico, moral y por la paciencia que tuvo. Quiero que sepan que los amo y que la persona que soy y hasta donde he llegado es gran mérito de ustedes.

En segundo lugar quiero agradecer a mi suegra a quien considero mi segunda madre. Siempre preocupada por mi alimentación, salud y estado de animo.

En tercer lugar y no por eso menos importante, a mi compañera de este y otros tantos viajes por venir, Nicole. Todas esas horas de trabajo junto a ti, sin hablarnos y molestarnos, pero siempre juntos. Por la paciencia que tuviste para aguantar las noches de trabajo, fines de semanas, y por escucharme en las noches de desvelo. Gracias por aguantar mis defectos, peticiones, enfados, etc. y a pesar de todo siempre darme tú apoyo y fuerza para completar este objetivo, el cual sin ti no habría sido posible. Quiero que sepas que eres un pilar fundamental para mí, te amo mi muñeca.

Por último agradezco a todos los que alguna vez tuvieron la preocupación y el interés de saber cómo estaba, haciendo las típicas preguntas, ¿Cómo va la tesis? o ¿Cuándo entregas la tesis?

RESUMEN

El presente trabajo de título tiene por objetivo diseñar un algoritmo genético paralelo para la resolución de problemas de optimización combinatoria. Para cumplir con el objetivo general es necesario contar con un problema de decisión complejo (NP-Hard), en este trabajo se selecciona el problema de localización de nodos presente en las redes inalámbricas. Una vez definido el problema a resolver se desarrollan los principales componentes del algoritmo genético como: la codificación utilizada, la función de fitness, los operadores genéticos, etc. Finalmente se realiza una comparación entre el algoritmo propuesto y su versión secuencial.

Para el desarrollo de este trabajo título se realiza una exhaustiva búsqueda bibliográfica, que permite tener una visión actualizada sobre las características y problemas presentes en las redes inalámbricas, los distintos tipos de algoritmos genéticos, los frameworks que permiten la implementación de metaheurísticas evolutivas y por último una descripción de la computación de alta performance.

El problema de localización de nodos se resolvió por medio de tres algoritmos genéticos, el primero de ellos es un algoritmo genético secuencial, el segundo es un algoritmo paralelo Maestro/Esclavo y el tercero es un algoritmo basado en un modelo de islas. Para los tres algoritmos desarrollados se implementaron la misma función objetivo y operadores genéticos.

Como los algoritmos genéticos son de naturaleza estocástica, para la comparación entre ellos es necesario realizar un elevado número de ejecuciones independientes sobre el problema planteado. Una vez obtenidos los resultados de las respectivas ejecuciones se realiza un análisis estadístico que permite verificar si los resultados obtenidos (fitness) provienen de poblaciones distintas o iguales. Luego se realiza una comparación del tiempo de procesamiento entre los distintos algoritmos utilizando el Speedup y la eficiencia.

Los algoritmos modelos de islas son los que obtienen los mejores resultados en función del tiempo de procesamiento. Respecto a los valores de fitness promedio obtenidos los algoritmos secuencial y los basados en modelos de islas son los que obtienen los mejores resultados.

Índice de Contenidos

Agradecimientos ii

Resumen iii

Índice de Contenidos iv

Índice de Tablas viii

Índice de Figuras ix

1 Introducción 1

1.1 Planteamiento del Problema 1

1.2 Objetivos del Estudio 2

1.2.1 Objetivo General 2

1.2.2 Objetivos Especifícos 3

1.3 Alcances o Ámbito del Estudio 3

1.4 Metodología Propuesta 3

1.5 Contenido del Trabajo de Título 4

2 Revisión Bibliográfica 6

2.1 Wireless Sensor Networks 6

2.1.1 Nodos Sensores 6

2.1.2 Topologías 7

2.1.3 Aplicaciones 8

2.1.4 Problemas de Optimización en WSNs 11

2.2 Problema de Estimación de Posición 13
2.2.1 Descripción del Problema 13
2.2.2 Métodos de Estimación de Distancia 14
2.2.3 Métodos de Estimación de Posición 16
2.3 Algoritmo Genético Secuencial 20
2.3.1 Analogía con la Evolución 20
2.3.2 Fundamentos 23
2.3.3 Elementos Básicos 25
2.4 Algoritmo Genético Paralelo 43
2.4.1 Taxonomía 44
2.4.2 Modelos Independientes 45
2.4.3 Modelo Maestro/Esclavo 45
2.4.4 Modelo Distribuido 46
2.4.5 Modelo Celular 48
2.4.6 Modelos Híbridos 48
2.5 Frameworks de Optimización con Metaheurísticas 49
2.5.1 Tipos de Frameworks 49
2.5.2 Framework ParadisEO 53
2.6 Computación de Alta Performance 55
2.6.1 Clasificación de las Arquitecturas 56
2.6.2 Arquitecturas MIMD Modernas 60
2.6.3 Métricas de Desempeño 62

3 Formulación del problema 65
3.1 Formulación 65
3.1.1 Problema de optimización 66
3.1.2 Restricciones del problema 68
3.2 Generación de Escenario 68

4 Algoritmo Propuesto y su Implementación 70
4.1 Algoritmo Genético Secuencial Propuesto 70
4.1.1 Representación de los Individuos 72

4.1.2 Operadores Genéticos 72
4.1.3 Función de Evaluación 76
4.1.4 Motor Evolutivo 78
4.1.5 Criterio de Convergencia 79
4.2 Algoritmo Genético Paralelo 79
4.3 Implementación computacional 82
4.3.1 Generación o Ingreso de Escenario 84
4.3.2 Datos y Parámetros 85
4.3.3 Datos de Salida y Despliegue de resultados 86

5 Pruebas y resultados **87**
5.1 Especificaciones de Hardware y Software 87
5.2 Diseño de Experimentos 88
5.2.1 Escenarios de Pruebas 89
5.2.2 Parámetros de Configuración 90
5.2.3 Configuración del Entorno de Pruebas 91
5.2.4 Análisis Estadístico 91
5.2.5 Análisis Empírico 93
5.2.6 Tipos de gráficos 93
5.3 Experimentos 94
5.3.1 Visualización Gráfica de los Datos 94
5.3.2 Metodología Estadística 96
5.3.3 Evaluación del Tiempo de Procesamiento 101
5.3.4 Métricas de Desempeño 103
5.3.5 Resultados para los Escenarios Propuestos 105

6 Conclusiones y Trabajo Futuro **120**
6.1 Análisis de los objetivos planteados 120
6.2 Trabajo futuro 122

Bibliografía **124**

Anexos **132**

A ParadisEO **133**

A.1 Instalación en Windows 133

A.1.1 Herramientas Necesarias 133

A.1.2 Instalación 134

A.2 Instalación en Linux Debian 134

A.2.1 Herramientas Necesarias 134

A.2.2 Instalación 135

A.3 Generando el primer proyecto 137

B Estructura de directorios **139**

C Archivo de Configuración **140**

D Archivo Datos de Entrada **141**

Glosario **143**

Índice de Tablas

2.1 Ventajas y desventajas en la utilización de MOFs (Parejo, 2013) 50
2.2 Lista de frameworks de Computación Evolutiva. 51
2.3 MOFs sobrevivientes. 53

5.1 Características de los escenarios. 89
5.2 Configuración Paramétrica (a)Algoritmo Genético Secuencial y Maestro/Esclavo; (b)Modelo de Islas. 91
5.3 Test de normalidad de Kolmogorov-Smirnov. 97
5.4 Test de Levene para la homogeneidad de varianzas. 97
5.5 Planteamiento de Hipótesis para el test de ANOVA. 98
5.6 Resultado del test de ANOVA para el escenario R4. 98
5.7 Resultado del test de ANOVA para el escenario R3. 99
5.8 Resultado de la prueba de Tukey para los escenarios R4 y R3. 99
5.9 Medias obtenidas para el escenario R3. 99
5.10 Planteamiento de Hipótesis para el test de Kruskal-Wallis. 100
5.11 Resultados del test de Krukal-Wallis para los escenarios R5 y R2. 101
5.12 Rendimiento de los algoritmos ($\overline{X}$ representa el tiempo medio en segundos, S_p es el *Speedup* y E_p es la *eficiencia*, donde p es la cantidad de núcleos). . . 104

Índice de Figuras

2.1 Arquitectura básica de un nodo sensor. 7
2.2 Topologías estrella y punto-a-punto. 8
2.3 Taxonomía aplicaciones Redes de Sensores, según (Yick et al., 2008). 9
2.4 Técnicas de estimación de distancia: (a) ToA un-sentido (b) ToA doble-sentido (c) TDoA 15
2.5 Comparación entre la biología y los algoritmo genéticos. 23
2.6 Estructura general de un algoritmo genético. 25
2.7 Ejemplo de una codificación. 26
2.8 Ejemplo de una codificación binaria. 26
2.9 Ejemplo de una codificación real. 28
2.10 Estrategias de la ruleta y Muestreo estocástico, según (Sivanandam & Deepa, 2008) 34
2.11 Estrategia de selección por torneo, según (Sivanandam & Deepa, 2008) . . . 35
2.12 Ejemplo de un operador de un punto de cruce. 37
2.13 Ejemplo de un operador de cruce uniforme. 38
2.14 Ejemplo de un cruce de tres padres. 38
2.15 Operadores de mutación para codificación binaria. 42
2.16 Estructura distribuida y celular de un AGPs, según (Luque & Alba, 2011). . . 44
2.17 Taxonomía de los algoritmos genéticos paralelos. 45
2.18 Ejecuciones independientes. 45
2.19 Modelo paralelo maestro-esclavo. 46
2.20 Modelos de migración. 48
2.21 Modelos mixtos, según (Luque & Alba, 2011) 49

2.22 Taxonomía de computación paralela, según (Tanenbaum, 2012). 55
2.23 Arquitectura Von Neumann 57
2.24 Arquitectura de Múltiples Instrucciones y Único Dato. 57
2.25 Arquitectura de Única Instrucción y Múltiples Datos. 58
2.26 Clasificación Johnson de sistemas de cómputo, según (Parhami, 2007). 59
2.27 Modelo típico multiprocesador de memoria compartida. 60
2.28 Modelo típico multicomputador de memoria distribuida. 60
2.29 Arquitectura de un multiprocesador simétrico con memoria caché. 62
2.30 Speedup de un programa paralelo, según (Talbi, 2009). 64

3.1 Falsa conexión entre nodos. 68

4.1 Solución codificada para el problema de localización, según (Molina, 2010) . 72
4.2 Operador single vertex neighborhood 75
4.3 Arquitectura de la solución propuesta. 80
4.4 Diagrama de flujo de las principales etapas de la implementación. 84

5.1 Escenario con distintos radios de comunicación. 90
5.2 Análisis estadísticos de los resultados experimentales. 92
5.3 Diagrama de cajas para los resultados de los distintos escenarios de pruebas. . 95
5.4 Tiempos de ejecución promedio para los escenarios R5 y R4. 102
5.5 Tiempos de ejecución promedio para los escenarios R3 y R2. 103
5.6 Speedup y eficiencia media obtenida por los algoritmos paralelos. 105
5.7 Comparación entre la topología real vs la encontrada por el algoritmo AGS para el escenario R5. 106
5.8 Comparación entre la topología real vs la encontrada por los algoritmos AGME-2, AGME-4 y AGisla-2 para el escenario R5. 107
5.9 Comparación entre la topología real vs la encontrada por el algoritmo AGisla-4 para el escenario R5. 108
5.10 Comparación del fitness promedio obtenidos por los modelos AGisla-2 y AGisla-4 para el escenario R5. 109
5.11 Comportamiento del mejor fitness y del fitness promedio para el escenario R5. 110

5.12 Comparación entre la topología real vs la encontrada por los algoritmos AGS, AGME-2 y AGME-4 para el escenario R4. 111
5.13 Comparación entre la topología real vs la encontrada por los algoritmos AGisla-2 y AGisla-4 para el escenario R4. 112
5.14 Comparación del fitness promedio obtenidos por los modelos AGisla-2 y AGisla-4 para el escenario R4. 113
5.15 Comportamiento del mejor fitness y del fitness promedio para el escenario R4. 113
5.16 Comparación entre la topología real vs la estimada por el algoritmo AGS para el escenario R3. 114
5.17 Comparación entre la topología real vs la encontrada por los algoritmos AGME-2, AGME-4 y AGisla-2 para el escenario R3. 115
5.18 Comparación entre la topología real vs la estimada por el algoritmo AGisla-4 para el escenario R3. 116
5.19 Fitness promedio obtenido por los modelos AGisla-2 y AGisla-4 para el escenario R3. 116
5.20 Comportamiento del mejor fitness y del fitness promedio para el escenario R3. 117
5.21 Comparación entre la topología real vs la encontrada por los algoritmos AGS, AGME-2 y AGME-4 para el escenario R2. 118
5.22 Comparación entre la topología real vs la encontrada por los algoritmos AGisla-2 y AGisla-4 para el escenario R2. 119

B.1 Estructura de directorio del programa. 139

C.1 Archivo de configuración para el modelo de islas. 140

D.1 Estructura del archivo “Anclas.txt”. 141
D.2 Estructura del archivo “Matriz.txt”. 142

CAPÍTULO 1

Introducción

1.1 Planteamiento del Problema

Los Algoritmos Genéticos (AG) son métodos genéricos aplicados, tradicionalmente, en problemas de búsqueda y optimización, utilizados para resolver problemas de alta complejidad (generalmente, donde la búsqueda exhaustiva o analítica no es factible). Típicamente, estos algoritmos funcionan de forma secuencial. Este tipo de Algoritmo Evolutivo intenta emular el proceso de evolución natural (mediante selección, cruzamiento y mutación de especies), el principal mecanismo que guía la aparición de estructuras orgánicas complejas y bien adaptadas.

Los avances de la computación, permiten hoy la ejecución de aplicaciones en estructuras paralelas. El paralelismo es una técnica de procesamiento basado en un principio aparentemente sencillo; "dividir un problema grande en grupos pequeños para resolverlos de forma simultanea". Gracias a este principio es posible la implementación de Algoritmos Genéticos Paralelos (AGP). Este concepto viene desarrollándose desde hace varios años, y es posible encontrar, hoy en día, diversas categorías que permiten reducir los tiempos de cómputo y, en algunas ocasiones, mejorar los resultados.

Los problemas de optimización combinatoria se caracterizan por tener grandes espacios de búsqueda, lo cual implica que pueden tener múltiples soluciones. Para poder obtener una

solución optima se debe contar con una gran cantidad de tiempo.

Los problemas de optimización combinatoria están relacionados comúnmente con problemas NP-Complete, lo cual significa que los recursos computacionales necesarios para encontrar una solución incrementan de forma exponencial con la magnitud del problema, especialmente cuando son tratados con algoritmos exactos.

Éste trabajo de título explorará en esta materia mediante el estudio, diseño e implementación de un AGP para la solución de problemas complejos de optimización. Al final del trabajo se espera contar con una plataforma computacional que permita realizar experimentación con distintos problemas de optimización combinatoria de la clase NP-Complete.

Se exploraran los distintos tipos de problemas de optimización combinatoria presentes en las redes de sensores inalámbricas (WSN), con el objetivo de utilizar uno de estos problemas para la evaluación del algoritmo genético paralelo. Dentro de los problemas catalogados como NP-Complete para las WSN se pueden encontrar, enrutamiento eficiente, optimización del tiempo de vida, estimación de la posición de los nodos, etc.

Para el diseño e implementación del AGP se realizará una búsqueda bibliográfica exhaustiva para conocer el estado actual de la temática, estudiando aplicaciones que utilicen este tipo de algoritmos. Finalmente, el algoritmo desarrollado será validado con el problema seleccionado, con el propósito de verificar su efectividad, tanto desde el punto de vista de los resultados, como del uso de los recursos computacionales.

1.2 Objetivos del Estudio

1.2.1 Objetivo General

- Diseñar e implementar una plataforma de algoritmo genético paralelo para resolver problemas de optimización combinatoria.

1.2.2 Objetivos Especifícos

- Seleccionar un problema de optimización combinatoria presente en las Wireless Sensor Networks.
- Diseñar y configurar los parámetros para el algoritmo genético paralelo.
- Encontrar posibles soluciones al problema planteado, utilizando el algoritmo genético paralelo.
- Comparar los resultados del problema entre un algoritmo genético secuencial y uno paralelo.

1.3 Alcances o Ámbito del Estudio

El lenguaje de programación utilizado para la implementación del algoritmo genético paralelo sera C++, sobre la plataforma Linux debian, además de utilizar alguna librería que cuente con herramientas para los algoritmos evolutivos. El algoritmo permitirá la configuración de los parámetros de entrada tanto del algoritmo genético como del problema de optimización combinatoria a resolver.

Para el desarrollo del algoritmo genético primero es necesario definir el problema a resolver, ya que es necesario conocer la función de evaluación y su codificación, para posteriormente implementar los operadores genéticos del problema.

Se evaluarán los problemas de optimización combinatoria presentes en las WSN con el objetivo de seleccionar el más indicado para la evaluación del algoritmo genético.

1.4 Metodología Propuesta

1. Se hará una revisión bibliográfica de los problemas de optimización combinatoria presentes en las Wireless Sensor Networks (WSNs), de modo de tener una visión actual del tema.

2. Seleccionar el problema a resolver.
3. Describir los tipos de Algoritmos Genéticos a través de una búsqueda bibliográfica.
4. Identificar los posibles framework disponibles para la implementación de algoritmos evolutivos.
5. Describir los tipos de computación de alta performance, definir las métricas de desempeño.
6. Definir el tipo de codificación que se utilizará para los cromosomas del AG.
7. Especificar la función de *fitness* que se utilizará para optimizar el problema.
8. Definir el criterio de selección de las parejas que se cruzarán para la obtención de las descendencias.
9. Estudiar los distintos tipos de operadores de cruce y mutación, con el fin de seleccionar la mejor combinación que permita encontrar una solución cercana al óptimo.
10. Evaluar mediante pruebas computacionales, el comportamiento del AG.
11. Validación del modelo y evaluación de las soluciones.

1.5 Contenido del Trabajo de Título

El presente trabajo de título está formado por 6 capítulos. En el capítulo 1 o "Introducción" se dan a conocer los objetivos generales y específicos, se presenta el alcance del estudio, y por último se presenta la metodología utilizada para realizar el trabajo de título.

En el capítulo 2 se realiza una revisión bibliográfica que presenta las principales características de las redes inalámbricas, como también, los tipos de problemas de optimización combinatorias presentes en las WSNs. Además, se dan a conocer los frameworks que cuentan con la capacidad de implementar algoritmos evolutivos, especialmente algoritmos genéticos paralelos. Por último, se presenta el marco teórico de la computación de alta performance.

En el capítulo 3, se describe la formulación matemática del problema de localización de nodos con sus respectivas restricciones. Se obtiene una formulación base para la solución del problema.

En el capítulo 4, se describen las etapas que son necesarias para la implementación de un algoritmo genético secuencial, como la representación de los individuos, operadores de mutación, cruce, selección y reemplazo. Además, se describe la implementación del algoritmo genético paralelo el cual hace uso de los mismos operadores descritos para el algoritmo secuencial. Por último, se expone la implementación computacional de los distintos algoritmos.

El capítulo 5 da a conocer los resultados estadísticos y métricas de desempeño obtenidas por los algoritmos propuestos, donde se verifica el comportamiento de éstos para los distintos escenarios de pruebas.

Finalmente en el capítulo 6 se presentan las conclusiones obtenidas para el trabajo desarrollado, y se proponen algunos trabajos futuros.

CAPÍTULO 2

Revisión Bibliográfica

2.1 Wireless Sensor Networks

Wireless Sensor Networks (WSNs) son un campo de investigación relativamente novedoso (Yick et al., 2008). Se encuentran constituidas por un gran número de dispositivos, con limitaciones de potencia y rendimiento, conocidos como nodos de sensores. Éstos ofrecen variadas posibilidades de sensado, cómputo y comunicación. Los nodos dentro de una WSN, si son considerados de forma individual, cuentan con capacidades limitadas, pero cuando actúan de forma colaborativa, permiten un gran espectro de posibilidades. El gran éxito de las WSNs ha permitido el desarrollo de nuevo hardware, múltiples capacidades de sensado, y una gran cantidad de nuevas aplicaciones. Junto con éstos avances, se cuenta con nuevas restricciones de diseño y objetivos de operación, es decir, una nueva gama de problemas de optimización novedosos y complejos (Dargie & Poellabauer, 2010).

2.1.1 Nodos Sensores

Los nodos sensores se pueden adaptar a los escenarios y aplicaciones para las que se utilizaran. La mayoría de los nodos sensores comparten muchas características, aunque siempre es posible encontrar un nodo para alguna aplicación específica. Sin embargo, esta tesis sólo se centra en los aspectos generales de las WSNs, por lo que, se describirán de forma general las propiedades más comunes entre los nodos sensores.

Un nodo sensor se caracteriza por tener un tamaño pequeño, un bajo costo, contar con uno o más sensores (humedad, temperatura, etc.), tener la capacidad de comunicación inalámbrica, capacidad de procesamiento, memoria y contar con baterías de alimentación. Se debe recalcar que las últimas cuatro características son las más limitadas.

La arquitectura básica de un nodo sensor, como se muestra en la Figura 2.1, se encuentra formada por una unidad de detección (contiene los sensores y el conversor análogo digital), la unidad de alimentación (almacena la energía disponible), la unidad de procesamiento (realiza el cálculo) y el transceptor (permite la comunicación inalámbrica).

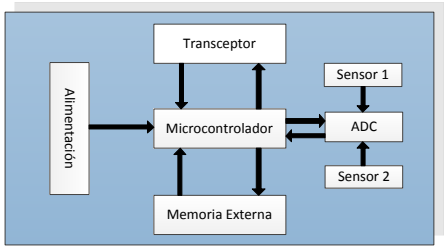


Figura 2.1 Arquitectura básica de un nodo sensor.

2.1.2 Topologías

Las WSNs se pueden formar en diferentes topologías dependiendo del tipo de red y aplicación. Algunas de las principales topologías pueden ser estrella, punto-a-punto y una combinación de las dos (ver Figura 2.2).

- Topología Estrella: en ésta topología un nodo se encuentra a cargo de un pequeño grupo de nodos (cluster) y de la comunicación desde y hacia su grupo. Los nodos y cluster se comunican a través de estos coordinadores y no directamente. Una de sus desventajas es la alta carga de información que debe soportar el nodo coordinador.

- Topología punto-a-punto: en ésta topología los nodos pueden comunicarse directamente con cualquier nodo que se encuentre dentro de su rango de comunicación. Si los nodos no se encuentran en el rango se puede redirigir la información a través de otro

nodo. Es más compleja de implementar, pero tiene como ventaja la disminución en la sobrecarga del coordinador.

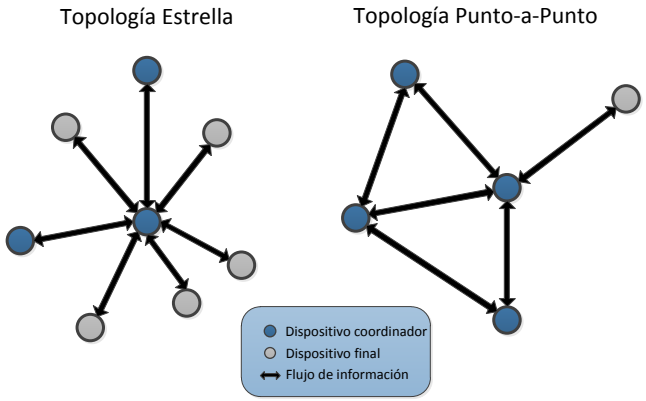


Figura 2.2 Topologías estrella y punto-a-punto.

2.1.3 Aplicaciones

Las áreas donde realmente es necesaria la aplicación de esta tecnología son principalmente las que implican un difícil acceso, ambientes hostiles, operación automatizada, espacios donde es necesario cubrir regiones amplias, etc. Algunos *survey* que describen las WSNs y sus aplicaciones se pueden encontrar en Yick, Mukherjee & Ghosal (2008); Yoneki & Bacon (2005); Arampatzis, Lygeros & Manesis (2005); García, Ibargüengoytia, García & Pérez (2007). Principalmente las aplicaciones se pueden dividir en dos categorías: monitoreo y seguimiento (ver Figura 2.3).

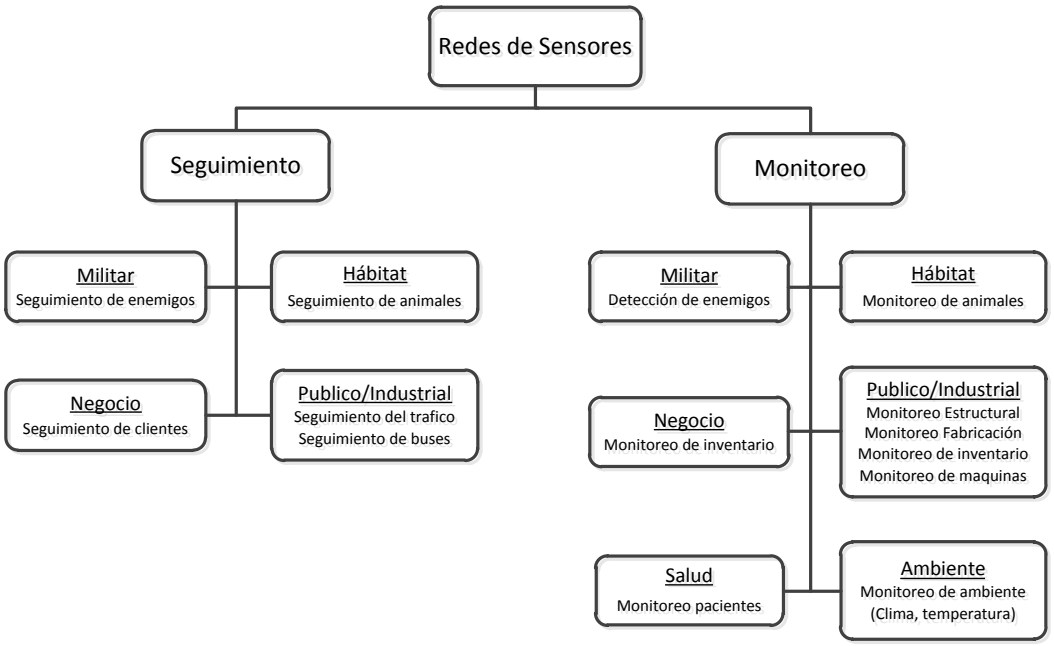


Figura 2.3 Taxonomía aplicaciones Redes de Sensores, según (Yick et al., 2008).

2.1.3.1 Aplicaciones Militares

Las aplicaciones militares han sido el principal motor en el desarrollo de las WSNs especialmente desde las etapas tempranas de investigación. Algunos de sus principales objetivos es el reconocimiento, detección y seguimiento de enemigos. En Lédeczi et al. (2005) se investiga y desarrolla un detector y localizador de francotiradores, para su desarrollo se distribuye una gran cantidad de sensores que tienen por objetivo detectar y medir el tiempo de llegada del *muzzle blasts* (sonido del disparo) y las ondas de choque del disparo. Los sensores envían sus mediciones a una estación base, como un ordenador, con el fin de estimar la ubicación del tirador utilizando un algoritmo. Este algoritmo esta formado por tres bloques principales, el primero de ellos es una codificación de cruce por cero, un detector de onda de choque y de muzzle blasts, ambos bloques miden el tiempo de arribo del evento.

2.1.3.2 Aplicaciones Publicas e Industriales

En éste tipo de aplicaciones lo más común es utilizar las WSNs en el monitoreo de alguna variable química o física de los procesos (humedad, calor, luz, presión, pH, etc.). También tiene usos en el área de la seguridad industrial, en Romero, Marín & Jiménez (2013) se presenta un sistema de monitoreo de alerta temprana de atmósferas explosivas, el principal objetivo es diseñar una red que sea capaz de recolectar la información necesaria sobre la concentración de gases. Además se puede implementar en la supervisión del estado de las máquinas pesadas (anticipación de fallas).

2.1.3.3 Aplicaciones de Salud

En estas aplicaciones se busca utilizar sensores pequeño, que no afecten el estilo de vida de los pacientes, generalmente se utilizan para monitorear el estado de los pacientes, por ejemplo, el monitoreo de infartos, alertas para personas sordas, el monitoreo de la presión arterial, etc. Para este tipo de aplicaciones los nodos sensores deben cumplir estrictos requerimientos de seguridad, y pueden ser utilizados de forma externa o interna (posicionados dentro del cuerpo del paciente). En Cárcamo & Bahamondes (2014) se utiliza una WSNs para el monitoreo de las actividades y frecuencia cardíaca de un paciente.

2.1.3.4 Aplicaciones Ambientales y Hábitat

Generalmente en las aplicaciones ambientales las WSNs se utilizan para la observación de las condiciones ambientales y sus variaciones (posiblemente por la intervención del hombre), también se utiliza para el seguimiento, monitoreo y recolección de datos fisiológicos de animales. Arce, Tech, Silva & Costa (2009) propone la implementación de una WSNs para generar una infraestructura que permita obtener datos biológicos de un rebaño de bovinos. Otro trabajo interesante se presenta en Zhang, Sadler, Lyon & Martonosi (2004), en el cual se desarrolla una WSNs móvil para seguir el comportamiento de los animales migratorios (en este caso las cebras).

2.1.4 Problemas de Optimización en WSNs

El amplio campo de aplicaciones para las WSNs y las nuevas técnicas desarrolladas para su implementación ha provocado la aparición de nuevos problemas de optimización y muchos de estos problemas pueden encontrarse dentro de la categoría NP-Complete, lo cual dificulta la obtención de soluciones precisas a través de los métodos tradicionales. Para la resolución de estos problemas algunos investigadores han propuesto Heurísticas, mientras que otros utilizan programación lineal, y por último se están utilizando Metaheurísticas. Algunos de los problemas más representativos para las WSNs se presentan en Nan & Li (2008) los cuales son: optimización de los recursos para mejorar el *lifetime*, estimación de la posición de los nodos, fusión multi-sensor, enrutamiento eficiente, y posicionamiento de nodos y optimización de layout.

2.1.4.1 Asignación de tareas

Este tipo de problema se puede describir como el proceso de asignación de tareas para los diferentes elementos de la red, definiendo el periodo de tiempo en el que cada nodo estará midiendo y comunicándose con el resto. Además de decidir la calidad del servicio que determinan el uso de los recursos, como la disipación de la energía y el ancho de banda para la comunicación.

2.1.4.2 Fusión Multi-sensor

La implementación de una red sobre una determinada región está limitada por la cantidad de energía y el número de nodos. Cada nodo sensor produce información de forma periódica a medida que monitorea su espacio. La función básica de una red de éste tipo, es que los datos detectados deben ser recogidos y transmitidos a una estación base o nodo base para su posterior procesamiento. Durante el proceso de recolección de datos, los sensores tienen la capacidad de unir los paquetes dentro de la red, con el objetivo de reducir el consumo de energía.

2.1.4.3 Enrutamiento Eficiente

La captura de datos por parte de los nodos sensores debe ser comunicada al *gateway*, es decir, los nodos deben tener la capacidad de responder las consultas del usurario. El proceso de transmitir la consulta del usuario al correspondiente nodo y como éste debe transmitir la información solicitada por parte del usuario es muy difícil de lograr. Esta comunicación se lleva a cabo por medio de un enfoque multi-salto (utilizando como puente a otros nodos sensores). Como los nodos tienen pequeñas dimensiones sólo pueden llevar una pequeña batería, por lo cual se necesita un algoritmo de enrutamiento eficiente que logre optimizar el uso de la energía y con esto lograr aumentar el tiempo de funcionamiento de la red.

2.1.4.4 Posicionamiento de Nodos y Optimización de layout

En éste tipo de problema se busca diseñar una estructura de comunicación eficiente entre los nodos. Debido a que los nodos sensores cuentan con energía limitada, un objetivo importante es reducir al mínimo el consumo de energía total. Por lo tanto, un diseño óptimo de la topología de la red puede dar lugar a un menor consumo de energía y lograr maximizar el tiempo de vida de la red.

2.1.4.5 Estimación de la Posición

El conocer la posición de los nodos es de gran importancia para el ruteo, programación, y la integración espacial de los datos obtenidos. Por ejemplo, en una implementación de detección de intrusos, es necesario saber dónde ha sido detectado el intruso, de igual forma, para la prevención de incendios es de mucha importancia conocer donde es detectado el fuego. Generalmente, una pequeña cantidad de nodos sensores (también conocidos como nodos baliza o anclas) tienen definida una posición absoluta por medio de GPS u otro método. El GPS es uno de los métodos más utilizados, pero no se puede replicar para todos los nodos presentes en la red, debido a su alto costo.

Finalmente para el desarrollo de esta tesis se hará uso del problema de estimación de posición de los nodos, con el fin de evaluar el comportamiento de los algoritmos genéticos propuesto.

2.2 Problema de Estimación de Posición

La localización de la información es fundamental para el buen funcionamiento de una WSNs; sin esta información muchas de las aplicaciones no podrían ser posibles. Normalmente el usuario no sólo necesita saber que está sucediendo, sino que también donde está sucediendo. Por ejemplo, en la vigilancia de un campo de batalla (Arora et al., 2004) conocer la ubicación del enemigo pude ser mucho más importante que solo saber si hay un intruso. Por otro lado, en muchas ocasiones para una buena administración de la red es necesario tener conocimiento sobre la posición de los nodos, como es el caso de los algoritmos de encaminamiento geográfico (determinan la forma en que los nodos descubren a sus vecinos y como seleccionan al mejor candidato para su siguiente salto) (Kim, Govindan, Karp & Shenker, 2005), que utilizan el conocimiento de la posición para mejorar la conectividad de la red y así ahorrar energía. Para obtener la posición de los nodos el GPS es el sistema de localización más conocido, aunque su uso está restringido por el entorno (por ejemplo, en el interior de edificios o en espacios con poca visibilidad no es posible su utilización) y su alto costo. Por éstos inconveniente se buscan nuevos métodos de localización como la auto-localización. En este tipo de método la colaboración entre los nodos es esencial para la obtención de las posiciones.

2.2.1 Descripción del Problema

El objetivo del problema de localización es obtener la posición geográfica absoluta o relativa de cada uno de los nodos presentes en la WSN. Uno de los métodos más utilizado para la obtención de la posición es proveer a cada nodo con un hardware de auto-localización, como es el caso del GPS “Global Positioning System”. Para una WSN con una pequeña cantidad de nodos y un entorno compatible, el uso del GPS es factible, pero en redes con una gran cantidad de nodos no es recomendado por un número de razones (Yoneki & Bacon, 2005):

- **Costos**: el GPS es un hardware de alto costo comparado con el bajo costo de un nodo sensor.
- **Consumo de Energía**: el GPS es un gran consumidor de energía.

- **Tamaño**: una de las ventajas de los nodos sensores son su pequeño tamaño y si se optara por agregarle un GPS su tamaño crecería en demasía.
- **Requerimientos**: el GPS tiene un conjunto de requerimientos para un buen funcionamiento, como el despliegue en exteriores y con líneas de visión directa a los satélites.

Para la estimación de la posición es necesario contar con dos pilares básicos, sin los cuales el problema no podría ser resuelto Molina (2010), estos son los nodos balizas y las referencias. Los nodos balizas, también conocidos como nodos anclas o *landmarks*, son un subconjunto de nodos que tienen la capacidad de conocer su propia posición desde un inicio. La posición se les puede definir de forma manual o utilizando algún hardware extra como el GPS (se utiliza cuando los nodos balizas están en movimiento). Por otro lado las referencias son tuplas de la forma $(n_i, n_j, \delta_{i,j})$, donde n_i y n_j son los nodos de la red y $\delta_{i,j}$ es la distancia estimada entre el par de nodos (n_i, n_j).

2.2.2 Métodos de Estimación de Distancia

Como se dijo en la sección 2.2.1 uno de los principales pilares son las referencias, dentro de éstas se encuentra uno de los principales datos, el cual es la distancia estimada $\delta_{i,j}$ entre dos nodos n_i y n_j. Existen varias técnicas con la cuales se puede obtener la distancia estimada, a continuación se describirán de forma breve (para mayor información consultar Dargie & Poellabauer (2010); Molina (2010)):

2.2.2.1 Tiempo de Arribo (ToA)

La distancia entre un nodo emisor y receptor puede ser determinada utilizando el tiempo de propagación de una señal a una velocidad conocida. Por ejemplo, si una señal viaja a la velocidad de la luz (300 km/s) tan solo demoraría 30 ns en recorrer una distancia de 10 m. Medir el tiempo que tardan en recorrer distancias pequeñas, señales que viajan a la velocidad de la luz, es una tarea compleja y costosa, ya que se necesitan osciladores de alta resolución.

Existen dos métodos para medir el tiempo de propagación el de un-sentido y el de doble-sentido, el primero calcula la diferencia entre la hora de salida y de llegada de la señal

(ecuación 2.1). Para éste método se requiere una buena sincronización entre los relojes de los nodos (Figura 2.4 (a)). En el segundo método el emisor es el encargado de medir el tiempo de ida y vuelta de la señal (Figura 2.4 (b)), la distancia es calculada con la ecuación 2.2.

$$D = (t_2 - t_1) \times v \tag{2.1}$$

$$D = \left((t_4 - t_1) - \frac{t_3 - t_2}{2} \right) \times v \tag{2.2}$$

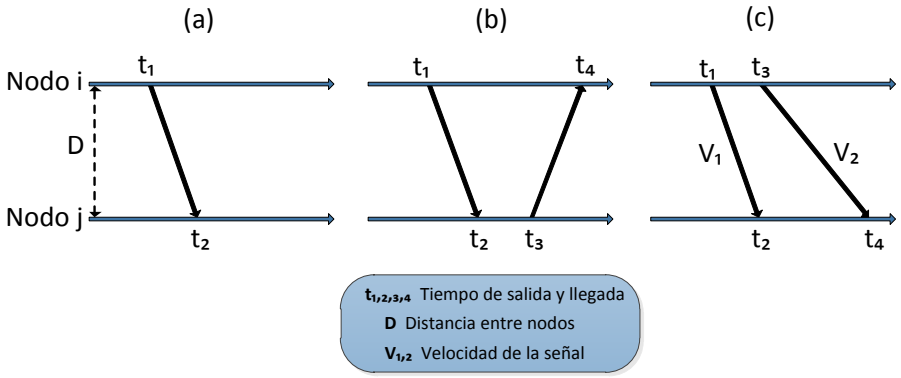


Figura 2.4 Técnicas de estimación de distancia: (a) ToA un-sentido (b) ToA doble-sentido (c) TDoA

Notar que en el método de un-sentido el nodo receptor calcula su posición, y en el segundo método el nodo emisor calcula la posición del nodo receptor.

2.2.2.2 Diferencia del Tiempo de Arribo (TDoA)

Éste método es una variación del método anterior, utiliza dos señales con diferente velocidad de propagación (Figura 2.4 (c)). La distancia puede ser calculada a partir del lapso de tiempo entre la primera señal recibida y la segunda, de la siguiente forma:

$$D = \left(\frac{t_4 - t_2}{t_3 - t_1} \right) \times \left(\frac{V_1 \times V_2}{V_1 - V_2} \right) \tag{2.3}$$

Donde V_1 y V_2 son las velocidades de la onda de radiofrecuencia (RF) y ultrasonido.

2.2.2.3 Fuerza de la Señal Recibida (RSSI)

Una señal que viaja por el espacio típicamente reduce su energía siguiendo una ley, la cual se puede modelar matemáticamente. Una de sus representaciones matemáticas más difundida se muestra en 2.4:

$$PL(d) = PL(d_0) + 10nlog\left(\frac{d}{d_0}\right) \tag{2.4}$$

Donde $PL()$ es la función perdida exponencial de la trayectoria medido en decibelios, d_0 es una distancia de referencia, y n es valor que depende del entorno.

Este es un método interesante ya que puede ser implementado en los nodos sensores sin la necesidad de utilizar algún hardware adicional. Sin embargo, es ampliamente sabido que es el método que presenta errores más significativos en el cálculo de la distancia, ya que es inestable y puede verse afectado por los obstáculos y el medio.

2.2.3 Métodos de Estimación de Posición

Luego de obtener las distancias entre nodos es necesario convertirla en coordenadas geográficas. Para lograr éste objetivo existen variadas técnicas. Algunas de éstas son: técnicas basadas en la geometría, escalamiento multidimensional, algoritmos de optimización, etc.

2.2.3.1 Algoritmos Matemáticos

En ésta categoría se encuentran las técnicas de trilateración, triangulación y multilateración. La primera calcula la posición de un nodo utilizando las distancias de él hacía tres nodos de referencias, en un espacio dimensional-2D se necesita por lo menos tres nodos de referencia. Para la técnica de triangulación se utiliza la distancia entre dos puntos de referencia y los ángulos de éstos hacia el nodo desconocido es similar a la técnica anterior pero con la diferencia que se utilizan ángulos. Finalmente para la multilateración se necesitan las distancias a un número determinado de balizas, para este caso la distancia se obtiene utilizando TDoA.

2.2.3.2 Escalamiento Multidimensional (MDS)

Para ésta técnica se utiliza una matriz de distancia entre cada par de nodos dentro de la red. Se aplican técnicas de descomposición espectral en la matriz de distancia para obtener una matriz de coordenadas de las posiciones relativas de cada nodo (Moreno, 2011). La técnica consiste en ordenar las distancias de cada par de nodo en orden decreciente según la distancia entre ambos, y buscar posteriormente de manera iterativa topologías que respeten ese orden. Ésta técnica puede alcanzar buenos resultados pero a un alto costo computacional.

En ésta técnica el problema de localización se puede representar como un problema de optimización de la siguiente forma:

$$\min_{x_1,\ldots,x_n} \sum_{i<j} (\|x_i - x_j\| - \delta_{i,j}) \qquad \text{y} \qquad \Delta := \begin{pmatrix} \delta_{1,1} & \delta_{1,2} & \cdots & \delta_{1,n} \\ \delta_{2,1} & \delta_{2,2} & \cdots & \delta_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ \delta_{n,1} & \delta_{n,2} & \cdots & \delta_{n,n} \end{pmatrix} \tag{2.5}$$

Dónde:

- x_i y x_j son las posiciones de los nodos i y j.
- $\delta_{i,j}$ es la distancia entre los nodos i y j.
- n es la cantidad de nodos.

2.2.3.3 Metaheurísticas

El problema de localización se puede formular como un problema de optimización. Se encuentra catalogado dentro de los problemas de optimización combinatoria del tipo NP-Complete (Bruck, Gao & Jiang, 2009; Feng, Girod & Potkonjak, 2006; Moore, Leonard, Rus & Teller, 2004). Al estar catalogado como un problema NP-Complete, significa que los recursos computacionales necesarios para encontrar una solución cuasi-optima se incrementan de forma exponencial con el tamaño del problema.

Para los problemas NP-Complete no se conoce un algoritmo exacto que permita encontrar soluciones en tiempo polinómico, pero existe la posibilidad de implementar técnicas aproximadas que permiten encontrar soluciones "cuasi-optimas", esto significa que las soluciones se encuentran cercanas al óptimo global (Pinedo, 2008). Dentro de los métodos aproximados se pueden encontrar las heurísticas y metaheurísticas.

Alcaraz, Maroto & Ruiz (2002) describe a las metaheurísticas como algoritmos de aproximación que se utilizan para resolver problemas de optimización combinatoria catalogados como difíciles, donde heurísticas clásicas no logran obtener resultados de una calidad aceptable. Las metaheurísticas están formadas por algoritmos nuevos que combinan diferentes conceptos derivados de heurísticas clásicas, inteligencia artificial, evolución biológica, sistemas neuronales y estadística.

Dentro de la literatura consultada las metaheurísticas más utilizadas para resolver el problema de localización son: recocido simulado (simulate annealign) y los algoritmos genéticos (genetic algorithm). A continuación se describen algunos puntos importantes de los trabajos analizados.

En Niewiadomska, Marks & Kamola (2011) se desarrollan tres algoritmos basados en recocido simulado, algoritmos genéticos y estrategias evolutivas, encontrando que éstos aumentan de forma considerable el rendimiento en comparación con técnicas basadas en programación lineal o cuadrática. Dentro de los tres algoritmos propuestos el que obtiene mejores resultados es el que utiliza en conjunto las técnica de recocido simulado y trilateración, aunque la basada en algoritmo genético no se encuentra muy lejos.

Zhang, Wang, Jin, Ye, Ma & Zhang (2008) propone un algoritmo genético con dos nuevos operadores genéticos. Su propuesta consigue obtener una mejor precisión en la optimización de la posición en comparación con un algoritmo basado en recocido simulado. Para la evaluación del algoritmo se utilizan cuatro topologías: malla uniforme, malla irregular, aleatoria uniforme y aleatoria irregular. Luego, dentro de estas redes se evalúa el comportamiento del algoritmo variando los rangos de comunicación entre los nodos y la cantidad de nodos anclas

utilizados.

Molina & Alba (2011) implementa tres metahurísticas las cuales son: recocido simulado, algoritmo genético y optimización por enjambre de partículas. Para el desarrollo de éstos algoritmos se utilizó una técnica de dos etapas, en la primera etapa se busca encontrar soluciones aproximadas utilizando como función de fitness la norma del error. Para la segunda etapa se utiliza una función de probabilidad, para la utilización de ésta función se debe tener conocimiento del problema. Finalmente, los resultados obtenidos en éste trabajo describen que el algoritmo recocido simulado obtiene mejores resultados que el algoritmo genético y el enjambre de partículas. Además, se sugiere la utilización de un 20 % de nodos anclas en la red, lo cual provee un buen balance entre precisión de localización y costo de equipamiento.

En Jiang, Jin, Guo & He (2013) se utiliza un algoritmo genético como método de localización. El algoritmo se divide en dos fases, en la primera etapa se calcula la distancia entre nodos, y en la segunda se ejecuta el algoritmo genético para la obtención de las posiciones. El algoritmo propuesto se compara con la técnica de escalamiento multidimensional (MDS), se observa que en algunos ambientes de evaluación la técnica MDS no es capaz de encontrar la posición de los nodos desconocidos. Por otra parte, de cinco topologías evaluadas solo en una MDS obtiene los mejores resultados, en la red donde los nodos son distribuidos en un espacio abierto. Finalmente, para la obtención de buenos resultado se recomienda entre un 15 % y 20% de nodos anclas.

Los algoritmos genéticos son ampliamente utilizados para la resolución de problemas de localización. Se pueden encontrar distintas formas de aplicación, por ejemplo, usando algoritmos genético como post-optimizador (Tam, yip Cheng & shan Lui, 2006; WANG, SUN & JI, 2013). Además, se puede complementar con otras técnicas, como el algoritmo de búsqueda de patrones (HUANG, CAI, WANG, SHE & MA, 2011).

De la literatura recopilada se concluye que la función de fitness a utilizar en este trabajo es 2.6, para la cual es necesario contar con la información de los nodos balizas y la matriz de distancia entre-nodos. En el capítulo 3 se describe de forma más detallada la formulación del

problema.

$$\min_{\substack{(x_i,y_i)\\ m<i\leq n}} \quad \sum_{i=m+1}^{n} \sum_{j\in N_i} \left| \sqrt{(x_i-x_j)^2+(y_i-y_j)^2} - \delta_{i,j} \right| \tag{2.6}$$

2.3 Algoritmo Genético Secuencial

2.3.1 Analogía con la Evolución

Los algoritmos genéticos intentan recrear el proceso de evolución natural. Con el fin de aclarar sus fundamentos, se describen las bases del origen de estos algoritmos.

Una de las investigaciones más influyentes para la actual teoría de la evolución, fue desarrollada por Darwin (1859), la cual ofrece una explicación a la diversidad biológica y como esta tuvo su origen a través de la *selección natural*, *adaptación al medio*, y la *herencia*.

La selección natural juega un papel importante en la evolución de los seres vivos. Pero uno de los pilares identificados por Darwin en su investigación, es la variación fenotípica (Fenotipo) que se encuentra presente entre los miembros de una población, Los rasgos fenotípicos son las características físicas y de comportamiento de un individuo que afectan directamente su respuesta al medioambiente. Cada individuo tiene su propia combinación de rasgos los cuales son puestos aprueba. Si se evalúan de forma favorable, se propagarán a través de la descendencia del individuo. De lo contrario, se desecharán al no poder dejar descendencia.

El fenotipo se puede definir en dos categorías: variables o constante. Así, el color de los ojos puede variar entre los individuos a colores Café, Azules o verdes, mientras que otras características, como el número de ojos con el que se nace, presenta un comportamiento constante.

En los algoritmos genéticos (AGs) se debe tener claro que el fenotipo representa el espacio o dominio donde se pueden encontrar las posibles soluciones. En caso de tener un

problema de optimización de una función cuyo dominio es un subconjunto de los números reales, entonces este subconjunto correspondería a los tipos de fenotipos posibles.

Uno de los principales problemas de la teoría de Darwin, fue la ausencia de un mecanismo válido para explicar la herencia. Gracias al desarrollo posterior de estudios en el área de la genética, por parte de Mendel, se logró dar respuestas a preguntas como: ¿Cómo se transmiten las características heredadas de una generación a la siguiente? ¿De qué manera se originan las variaciones sobre las cuales actúa la selección natural?.

La vista microscópica de la evolución natural es ofrecida por la *genética molecular*. Esta rama de la genética pretende dar a conocer los procesos que suceden en la representación de las características fenotípicas visibles de los individuos, en particular las relacionadas con la herencia. La observación fundamental de la genética es que cada individuo tiene una naturaleza dual; sus propias características fenotípicas se encuentran representadas a nivel genotípico (Genótipo). En otras palabras, el genotipo de todos los individuos codifica su fenotipo. Esta codificación no es uno a uno; un gen podría afectar las características fenotípicas, pero a su vez, un rasgo fenotípico puede ser determinado por más de un gen. Las variaciones fenotípicas siempre son causadas por el genoma, que a su vez es alterado por consecuencia de la mutación de genes o por la recombinación de genes en la reproducción sexual (Eiben & Smith, 2003).

El Genoma es toda la información genética de un ser vivo que contiene las instrucciones referente al desarrollo de un individuo especificando a grandes rasgos el lugar al que pertenece en el orden natural. Un organismo es un programa abierto que requiere de información extra que no está codificada en el genoma. Esta información no genética, conocida como el componente ambiental, tiene un grado elevado de impredecibilidad, por lo que el estado de un organismo en un momento dado depende tanto de su genoma como de la sucesión de acontecimientos que forman la historia vital del individuo (Eiben & Smith, 2003).

Todos los organismos vivos están formados por células, y cada célula contiene el mismo conjunto de uno o más Cromosomas, que a la misma vez están formados por ADN, el cual

sirven como una "guía" para el funcionamiento del organismo. Un cromosoma puede ser conceptualmente dividido en genes, cada uno de los cuales codifica una proteína particular. A grandes rasgos, se puede pensar en un gen como el encargado de codificar un rasgo, como el color de los ojos. Los diferentes "ajustes" posibles de un rasgo (por ejemplo, azul, café, verde) se llaman Alelos. Cada Gen está situado en un Locus en particular (posición) en el cromosoma (Mitchell, 1998).

El Código genético en los algoritmos genéticos, es la codificación que permite explorar el dominio del problema de una forma cómoda. Cada configuración de estas estructuras constituye el equivalente al cromosoma de un individuo en términos biológicos, y el conjunto de cromosomas forman el genotipo. Es frecuente que el código de los elementos del dominio del problema utilice una cadena de números binarios (ceros y unos), sin embargo se pueden utilizar representaciones en números enteros o cadenas de caracteres. En una representación binaria los bits individuales o bloques cortos de bits adyacentes serían el equivalente a los genes que codifican un elemento particular de la solución candidata. Un alelo es el posible valor que puede tomar un gen. Por ejemplo, en una cadena de bits los valores que se pueden seleccionar son 0 o 1; para alfabetos más grandes más alelos son posibles en cada locus (es la posición dentro de un cromosoma) (Mitchell, 1998), en la Figura 2.5 se puede apreciar una equivalencia de la terminología que se usa en los AGs respecto a la usada de forma habitual en la genética biológica.

Algunas de las características principales que se deben tener en cuenta de la evolución, están descritas por Davis (1991) citado por Ramos (2004):

- Las poblaciones de individuos se diferencian entre ellas por contar con distintas propiedades y habilidades.
- La población está limitada a una cierta cantidad de individuos.
- Los cromosomas que generan estructuras con éxito, tienen una mayor probabilidad de reproducción.
- Las mutaciones pueden causar que los genes de los hijos sean diferentes al de los padres.

- La combinación de material genético entre ambos padres puede generar configuraciones de genes muy diferentes.
- La evolución biológica no puede retractarse de los cambios generados.

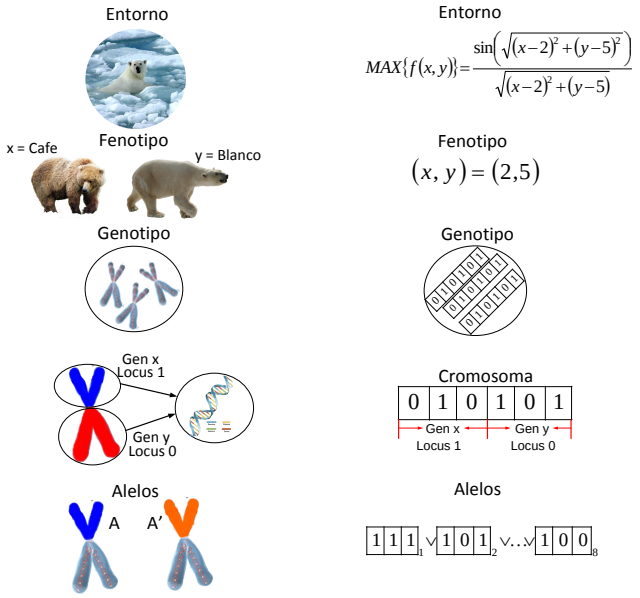


Figura 2.5 Comparación entre la biología y los algoritmo genéticos.

2.3.2 Fundamentos

Los algoritmos genéticos cuentan con una población de individuos que realizan una búsqueda en múltiples direcciones de forma paralela. Tiene por objetivo mantener a las potenciales soluciones y dar prioridad al intercambio de material genético entre los mejores individuos. Estos algoritmos simulan el comportamiento de la evolución biológica, donde por cada generación se generan nuevos individuos (hijos) que heredan características de los padres.

Para la resolución de problemas de optimización haciendo uso de los algoritmos genéticos, es necesario definir ciertos puntos que son fundamentales para el buen funcionamiento de éstos:

- Es necesario definir una representación adecuada para los individuos, generalmente es un vector con una codificación binaria, real o caracteres.
- La población inicial se puede generar de forma aleatoria o utilizando alguna heurística en particular.
- La función de fitness es de suma importancia, ya que de ésta dependerá la calidad de los individuos, y por ende definir cuales son los que mejor se adaptan al medio.
- Para la selección de los mejores individuos y la creación de descendencia es necesario crear operadores genéticos (selección, reemplazo, cruce y mutación), estos operadores dependerán de la codificación utilizada.
- Las probabilidades de cruce y mutación definen el porcentaje de individuos que serán seleccionados para los operadores de cruce y mutación.
- Tamaño de la población.
- Para el criterio de parada se puede utilizar el total generaciones o el fitness alcanzado.

En la Figura 2.6 se describe de forma general las etapas de funcionamiento de un algoritmo genético. Una vez definidos y diseñados todos los parámetros expuestos anteriormente, se comienza con una población generada de forma aleatoria, cada individuo generado representa una posible solución. Los individuos son evaluados con la función de fitness definida para el problema. Posteriormente, se comienza a iterar, y por cada iteración se genera una población a partir de la anterior, haciendo uso de los operadores genéticos de selección, cruce, mutación y reemplazo. Finalmente, dependiendo del criterio de parada utilizado el algoritmo arrojara al mejor individuo de la última población encontrada.

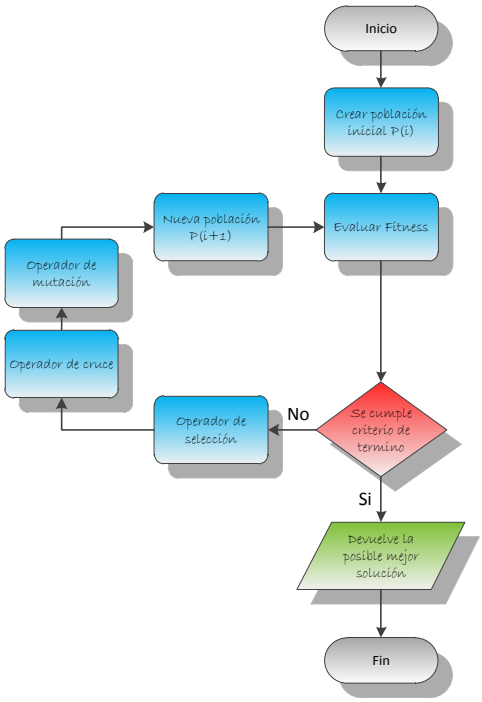


Figura 2.6 Estructura general de un algoritmo genético.

2.3.3 Elementos Básicos

2.3.3.1 Codificación de los Individuos

Dependiendo del tipo de problema, el algoritmo genético puede no operar directamente sobre las soluciones, sino sobre una representación codificada de éstas, parecido al material genético de un individuo (ver Figura 2.7). La codificación es de suma importancia, ya que de ésta dependerá el resto de etapas que se deban diseñar.

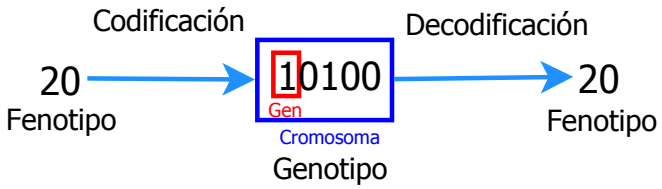


Figura 2.7 Ejemplo de una codificación.

Existen varias representaciones para la codificación del fenotipo. A continuación se discuten dos tipos básicos de codificación, binaria y real (punto flotante).

La codificación binaria es una de las más utilizadas debido a que es la propuesta originalmente por Holland (1975), consiste en asignar a un vector valores de ceros y unos. Dependiendo de la precisión que se necesite para el problema se define el tamaño máximo del vector. Por ejemplo, si se tiene una función objetivo que se debe maximizar y su dominio se encuentra entre los valores reales de 0 a 255, las soluciones se pueden codificar en una cadena binaria de ocho bits (ver Figura 2.8).

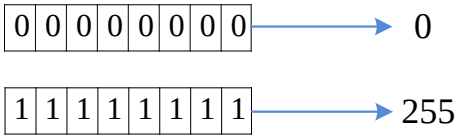


Figura 2.8 Ejemplo de una codificación binaria.

Este tipo de representación presenta ventajas en problemas que cuentan con una cantidad moderada de variables y no requieren demasiada precisión en sus resultados. Aunque se puede utilizar para problemas que cuentan con muchas variables, para ésto es necesario aumentar el tamaño de la población y la longitud de los cromosomas, lo cual implica un aumento considerable en el tiempo de cálculo (Thakur, 2014). Un ejemplo sería si se quisiera representar cada punto de un número decimal (en base 10), cada número necesitaría 4 bits para su representación. Por lo tanto para una variable con ocho cifras significativas se necesita $8x4 = 32$ bits en el gen. Si hubiera diez variables, significaría que un cromosoma debe

contener 320 bits. Tener un gen largo retrasa la convergencia del AGs, por lo tanto, solo se debe representar las variables con la precisión mínima necesaria.

Para muchos problemas, una codificación binaria puede no ser adecuada debido a que puede presentar los problemas descritos por Dasgupta & Michalewicz (1997), los cuales son la epístasis, representación natural y soluciones infactibles. La *epístasis*, se presenta cuando un gen tiene la capacidad de suprimir la contribución de los otros genes en el valor del fitness, si los genes contribuyen de forma independiente entonces se tiene una baja epístasis, pero si la contribución en la aptitud de un gen depende de los valores de los otros genes, el sistema tiene una alta epístasis. La *Representación natural* se presenta cuando el problema a resolver requiere de una representación de un orden más alto que el binario, un alfabeto de orden superior (ej. utilizar el fenotipo como genotipo de forma directa). Por ultimo las *soluciones infactibles*, se pueden presentar cuando se aplican los operadores genéticos, ya que una codificación binaria puede no tener la capacidad de representar un punto del espacio de búsqueda (landscape).

Otro inconveniente en la codificación binaria es el problema de "Hamming-Cliff", se presenta cuando la distancia de Hamming (en la codificación binaria) entre dos números enteros adyacentes (en código decimal) es muy grande. Por ejemplo, si se codifican los números 511 (0111111111) y 512 (10000000000), y luego se requiere pasar de un número a otro, se necesitan realizar 10 mutaciones lo que implica que la distancia de Hamming es muy grande. Esto significa que mientras más grande sea la distancia, menor es la posibilidad de que un individuo valido se transforme en un nuevo individuo valido.

Para solucionar el problema de la distancia de Hamming se propuso una nueva forma de codificación, la cual es conocida como codificación gray, su principal característica es que la separación que existe en el espacio codificado entre soluciones adyacentes en el espacio decodificado siempre es 1.

En la década de los noventa, se comienzan a investigar nuevas formas de representación, como la codificación de números con punto flotante. El objetivo de este tipo de representación

es estar más próximo al espacio de búsqueda. Esto tiene por consecuencia que los operadores genéticos sean específicos al problema a resolver. Por ejemplo, dos individuos cercanos en el espacio de codificación, también serán cercanos en el espacio del problema (Salto, 2000). Ésta representación hace uso de un vector de tamaño determinado por la cantidad de genes con los que está formado y cada gen es un número real (ver Figura 2.9).

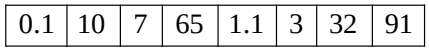


Figura 2.9 Ejemplo de una codificación real.

En Janikow & Michalewicz (1991) y Michalewicz (1994) se presentan diversos experimentos comparando AGs con representación real y binaria, concluyendo que la mejor opción es utilizar una representación real para problemas de optimización que cuenten con una función objetivo de dominio continuo, debido a que ofrece una mayor consistencia, precisión (especialmente cuando una representación binaria requiere una estructura prohibitivamente larga) y velocidad durante su ejecución.

A pesar de que puede traer consigo muchos beneficios, Thakur (2014) describe que se debe tener especial cuidado en el tipo de operador de cruce que se utilice, debido a que si no se usa un apropiado operador de cruce, es posible que se produzca el problema de convergencia prematura.

Dependiendo del tipo de problema se debe utilizar una codificación en particular, en la literatura existen muchas otras estrategias de codificación distintas a las nombradas, las cuales pueden ser representación matricial (Suárez et al., 2013; Larranaga et al., 1999), codificación por permutación (Salto, 2000; Ramirez, 2010) y caracteres.

2.3.3.2 Tamaño de la Población

La selección adecuada de un tamaño de población es otro de los puntos importantes a considerar en un AGs, ya que si ésta es muy pequeña, el algoritmo podría converger prematuramente

y por lo tanto no encontrar una buena solución, y sí es muy grande quizás se desperdiciarían muchos recursos computacionales al gastar mucho tiempo en mejorar los individuos. Este parámetro es uno de los factores más importantes que determinan el tiempo total de ejecución de un algoritmo genético (González, 2011). Típicamente la selección de este parámetro se realiza de forma manual, probando distintos tamaños hasta que se logre encontrar una que presente las mejores soluciones, aunque existen investigaciones (Liao & Sun, 2001) que sugieren utilizar la formula 2.7, donde *pob* es el tamaño de la población y $length$ es el largo del cromosoma. Desde las primeras propuestas en Holland (1975) para la selección del parámetro hasta nuestros días, las poblaciones de tamaño fijo siguen siendo las que predominan, como lo indican Eiben & Smith (2003), a pesar de que esté parámetro es de suma importancia para un óptimo funcionamiento. Harik, Cantú-Paz, Goldberg & Miller (1999) realizaron uno de los análisis más rigurosos para determinar un tamaño apropiado, con el objetivo de lograr un equilibrio entre la eficacia y la eficiencia de los algoritmos, aunque esta investigación trajo nuevas ideas en la selección adecuada del parámetro, se encuentra lejos de resolverlo.

$$pob = 1,65 \cdot 2^{0,21 \cdot length} \tag{2.7}$$

Durante los últimos años se han propuestos nuevos métodos para seleccionar un tamaño apropiado, las cuales se enfocan en utilizar estrategias de población de tamaño variable. En Ma & Krings (2008) se realizó una comparación entre varios tipos de poblaciones dinámicas (aleatorias, aumentando, disminuyendo y con forma de campana) y estáticas, los experimentos encontraron que las poblaciones dinámicas mejoran entre un 38 % a 50 % en comparación con las de tamaño fijo, éstas mejoras suceden en la cantidad de evaluaciones del fitness y los requisitos de espacio de memoria. Además se concluye que si el espacio de búsqueda es amplio es recomendable comenzar con una población de gran tamaño. Por último existen otros documentos donde se sugiere utilizar la técnica de poblaciones caóticas para generar poblaciones dinámicas (Ma, 2012).

2.3.3.3 Población Inicial

Existen dos formas de generar los individuos de la población. De forma aleatoria (la más utilizada en los AG) haciendo uso de los generadores pseudo-aleatorios (RNGs de forma abreviada), o aplicando algún algoritmo heurístico cuasi-aleatorio. Las ventajas del primero,

son la rapidez y la diversidad de las soluciones generadas, aunque puede tener el inconveniente de crear una población a la cual le puede costar mucho trabajo llegar a encontrar buenas soluciones. El segundo método presenta mejores soluciones, por lo que le permite converger de forma más rápida, pero se debe evitar generar una población con poca diversidad, ya que puede provocar que el algoritmo converja de forma prematura al quedar atrapado en un óptimo local Maaranen, Miettinen & Mäkelä (2004); Alcaraz, Maroto & Ruiz (2002).

2.3.3.4 Función de Evaluación

La función de evaluación en conjunto con la codificación son de suma importancia para todo algoritmo evolutivo. Ésta es responsable de determinar qué individuos dentro de una población son los más aptos para encontrar una solución a un problema en particular. Para una población de cromosomas, la función asigna a cada individuo un valor de fitness, el cual indica que individuo tiene mayores posibilidades de ser elegido para la siguiente generación. Gran parte del tiempo que necesita un AGs es utilizado por la evaluación de fitness, ya que se debe evaluar cada individuo de la población, por éste motivo se tiene que definir una función que no sea demasiado compleja y permita discriminar de forma correcta la calidad de los individuos.

Dependiendo del tipo de problema que se intente resolver, según Talbi (2009) la interpretación de las funciones de evaluación se pueden obtener directamente de la función objetivo formulada originalmente. Para otros casos la función objetivo se tiene que transformar con el propósito de obtener una función de evaluación que tenga mejor convergencia.

En la construcción de una buena función de evaluación se deben considerar algunos criterios. Malan & Engelbrecht (2013) describe entre los criterios más importantes el grado de interdependencia de las variables y el ruido.

El grado de interdependencia está relacionado de forma directa con la epístasis, y se refiere al grado de dependencia entre los genes de un cromosoma. Cuando las variables en un problema de optimización son dependientes entre sí, significa que no es posible sintonizar una variable de forma independiente de las otras para encontrar una solución óptima. Por

ejemplo, si las diferentes variables en una expresión matemática de una función de aptitud están separadas por sumas, las variables contribuyen de forma independiente a la aptitud, sin embargo, si se combinan diferentes variables a través de la multiplicación, entonces estas variables deben cooperar a fin de contribuir a la aptitud, si alguna variable tiene un valor bajo, entonces el producto puede ser bajo, incluso si la otra variable tiene un valor alto. Por último gracias a los estudios de Salomon (1996) y Czarn, MacNish, Vijayan & Turlach (2007) citados por Malan & Engelbrecht (2013), se ha demostrado que las funciones linealmente separables son más fáciles de resolver para los AGs que las funciones no-linealmente separables.

Malan & Engelbrecht (2013) describe que una pequeña cantidad de ruido podría ayudar a los algoritmos a tener un mejor funcionamiento en comparación con la ausencia de éste. Si se quiere detectar el ruido en la función de evaluación se pueden volver a muestrear los datos, para luego aplicar alguna medida de diferencia como la varianza o desviación estándar.

Algunos de los problemas que afectan el buen funcionamiento de los AGs son la existencia de gran cantidad de óptimos locales, y que el óptimo global esté muy alejado. Para el diseño de una buena función de fitness ésta debe tener la capacidad de representar los valores de los individuos de forma real. Gran parte de los problemas de optimización, cuentan con muchas restricciones o su espacio de búsqueda es de gran tamaño, por lo que muchos de los individuos pueden ser considerados como inválidos.

Cuando la función objetivo cuenta con restricciones especificas para el problema planteado, se propone el uso de algunas técnicas para el manejo de éstas, como se propone en Talbi (2009). La primera técnica se denomina estrategia de rechazo, tiene por objetivo conservar durante la búsqueda, sólo las soluciones factibles y las no-factibles se descartan de forma automática, o bien se les asigna un valor de fitness igual a cero. Este tipo de estrategia es práctica sólo si la población de soluciones no-factible es pequeña.

Otra posibilidad consiste en reconstruir a los cromosomas que no cumplen con las restricciones. Consiste en una heurística que cuenta con la capacidad de corregir los individuos

no-factibles. Generalmente son conocidos como operadores de reparación, y son específicos para cada problema.

Desafortunadamente no siempre es fácil "reparar" a los individuos, y a veces no es posible encontrar una codificación que evite soluciones no factibles. Para este caso el enfoque clásico es el uso de estrategias de penalización, para éste método las soluciones no factibles son consideradas durante el proceso de búsqueda. Esta solución consiste en reemplazar la función de evaluación por una función de penalización que incluya la función de fitness original más las restricciones del problema, el fitness obtenido por los individuos tendrá relación con las restricciones violadas o bien con el costo de reparación. Si la penalidad es lo suficientemente grande, individuos altamente inviables raramente serán seleccionados para la selección, y el AGs se concentrará en soluciones factibles o cuasi factibles. Para mayor información con respecto a los tipos de penalización ver Gen & Cheng (1996) citado por Talbi (2009) y Lin (2013).

En muchos problemas de optimización, la función de evaluación puede ser muy compleja. Talbi (2009) y Haupt & Haupt (2004) proponen que para reducir el tiempo necesario en el calculo de una función de fitness compleja, existe la opción de utilizar una aproximación de la función original que permita obtener una población con valores cercanos entre ellos, para que luego se reemplace por la función original. Este enfoque es conocido como meta-modelado. Existe un compromiso entre la complejidad del modelo y su precisión. La construcción de múltiples funciones locales en lugar de una global puede ser beneficioso.

Durante la ejecución del algoritmo existe la posibilidad de que se generen cromosomas idénticos, cuando sucede esto, Michalewicz (1994) sugiere que cada vez que se genera un nuevo cromosoma se compare con los individuos de la población actual con el fin de descartarlo si existe uno idéntico, este método se recomienda cuando la función de evaluación es muy compleja y necesita un tiempo mayor que el de hacer una comparación entre los individuos.

Otro método para disminuir el tiempo de cálculo de la función. Es solo evaluar la aptitud

de la descendencia de los cromosomas mutados y cruzados. Los padres no modificados ya tienen una aptitud asociada, por lo que no es necesario que sean evaluados nuevamente.

2.3.3.5 Operadores de Selección

Una vez definido el tipo de codificación y función de evaluación a utilizar, el siguiente paso es decidir cómo se llevara a cabo la selección, es decir, cómo elegir a los individuos de la población que crearán descendientes para la próxima generación y el número de hijos que tendrán. El objetivo de éste operador tiene como fin decidir cuáles son los individuos más aptos de la población.

La selección es el operador que se encarga de seleccionar a los cromosomas de una población, todo esto de acuerdo a los valores de aptitud. Mientras mejor sea la aptitud de un individuo más posibilidades tiene de ser seleccionado. La probabilidad de selección (o supervivencia) se define como el grado en que los mejores individuos son favorecidos. Cuanto mayor es este valor, los mejores individuos tendrán una mayor preferencia, pero existe un alto riesgo de que el algoritmo converja de forma prematura, y se encuentren solución incorrectas (óptimos locales). Por el contrario, si la probabilidad es muy baja, el algoritmo convergerá lentamente, y por lo tanto al algoritmo le tomara más tiempo encontrar una solución aceptable (Sivanandam & Deepa, 2008).

Según Talbi (2009) la asignación de la aptitud para cada individuo puede ser de dos formas diferentes:

- Función de selección proporcional a la función de evaluación, a cada individuo se le asigna una aptitud absoluta obtenida por la función de evaluación.
- Asignación de la aptitud proporcional al rango de los individuos, una aptitud relativa es asociada a los individuos. Por ejemplo, se crea un rango para la población el cual es asociado a cada individuo de acuerdo a su posición en una clasificación decreciente.

Una vez definida las formas de asignar la aptitud a los individuos se debe proceder a seleccionar a los padres de acuerdo a su fitness por medio de los siguientes métodos (Talbi,

2009): selección por rueda de la ruleta, muestreo universal estocástico, selección por torneo y selección basada en el rango.

La estrategia de *selección por ruleta* es la más común, se basa en asignar a cada individuo una probabilidad de selección proporcional a su aptitud relativa. La probabilidad de ser seleccionado es $p_i = \frac{f_i}{(\sum_{j=1}^{n} f_j)}$, donde f_i es la aptitud del individuo p_i en la población P. Se genera una "ruleta" virtual basándose en el fitness de cada uno (Figura 2.10(a)). Así, los peores individuos tendrán una menor porción de la ruleta que los mejores individuos, luego se generan números aleatorios que simulan posiciones dentro de la ruleta, y se eligen los individuos correspondientes a dicha posición. Los mejores individuos tienen un mayor espacio en la ruleta, por lo que tienen mayores probabilidades de ser seleccionados.

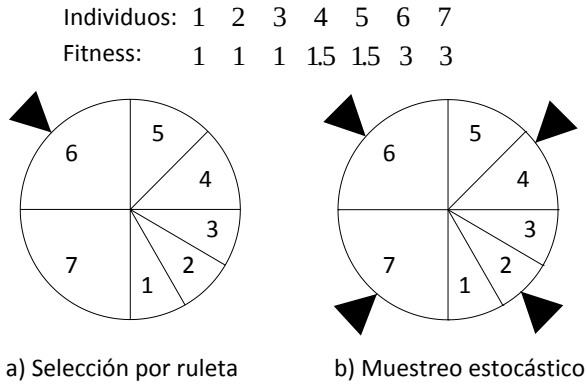


Figura 2.10 Estrategias de la ruleta y Muestreo estocástico, según (Sivanandam & Deepa, 2008)

En el *muestreo universal estocásticos* los individuos son asignados a los segmentos de una ruleta (Figura 2.10(b)), de tal forma que el tamaño de éste es igual al tamaño de su aptitud, de igual forma que el método de la ruleta. Utiliza un único giro de la ruleta para seleccionar a los individuos igualmente espaciados y con comienzo aleatorio. Si tenemos N_p números de individuos a seleccionar, entonces la distancia entre los punteros sera $\frac{1}{N_p}$ y la posición del primer indicador estará definido por un número aleatorio seleccionado del rango $[0, \frac{1}{N_p}]$.

La estrategia de *selección por torneo* está basada en el "elitismo". La forma más común de torneo es elegir p individuos al azar de la población y seleccionar el mejor de todos ellos (Figura 2.11). El tamaño del torneo es p, y la elección de ese parámetro es un factor importante, ya que si es grande favorecerá a los mejores individuos de toda la población y nunca se elegirá a los no tan buenos, mientras que si es pequeño la probabilidad de elegir a un individuo malo será bastante alta (Hsiao-Lan, 1994). Según (Sivanandam & Deepa, 2008) éste es un método eficiente y generalmente conduce a soluciones cuasi-óptima.

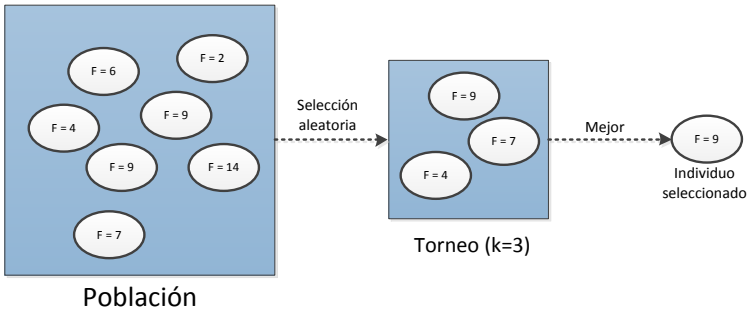


Figura 2.11 Estrategia de selección por torneo, según (Sivanandam & Deepa, 2008)

Selección basada en el rango, define a la probabilidad de selección en base a la aptitud relativa. Ordena a los individuos en base a sus valores de fitness de forma creciente (el individuo con mejor fitness tiene la posición más alta), luego para cada individuo se asigna una probabilidad de selección de acuerdo a su posición (en lugar de hacerlo de acuerdo a su fitness real). Para definir el rango se puede utilizar el mapeo lineal o exponencial. Según describe Sivanandam & Deepa (2008) la convergencia es lenta, pero impide la convergencia prematura. Además mantiene la probabilidad de selección cuando la varianza de la aptitud es baja. Conserva la diversidad y por lo tanto puede conducir a una búsqueda exitosa.

2.3.3.6 Operadores de Cruce

El operador de cruce permite el intercambio de genes entre los cromosomas de la población correspondiente, por lo que puede tener un gran impacto en el funcionamiento del algoritmo, es uno de los principales responsable de la exploración y explotación del espacio de búsqueda

(Coello et al., 2007), pero debe existir un equilibrio con el operador de mutación. Es el operador que ha recibido mayor atención por parte de los investigadores. Estos operadores mezclan genes de distinto individuos (llamados padres), para formar dos hijos que cuenten con la herencia de sus progenitores, aunque existe la posibilidad de cruzar más de dos individuos. La cantidad de individuos que se cruzaran está definida por la probabilidad de cruce. Si la probabilidad de cruce es 100%, entonces toda la generación siguiente es creada por el operador de cruce. Si es 0%, la nueva generación se genera a partir de copias exactas de los cromosomas padres (pero esto no significa que la nueva generación sea la misma). El cruce se realiza con la esperanza de que los nuevos cromosomas contengan los mejores genes de los padres y por lo tanto los nuevos cromosomas sean mejores (Sivanandam & Deepa, 2008). El cruce se realiza, por lo general, después de aplicar el operador de selección y depende del tipo de codificación utilizada González (2011).

Algunos criterios importantes que se deben considerar en el diseño o el uso de un operador de cruce son descritos en (Sivanandam & Deepa, 2008):

- La principal característica del operador de cruce es la heredabilidad. A través del operador genético los hijos deben heredar material genético de los padres.
- El operador de cruce debe producir soluciones válidas. Esto no siempre es posible en los problemas de optimización con restricciones.

Éste operador cuenta con tres etapas principales, el primero de ellos es seleccionar a dos o más individuos que jueguen el papel de padres. Luego se debe seleccionar un punto de cruce de forma aleatoria a lo largo del cromosoma. Y por último los valores que se encuentran antes y después del punto de cruce se deben intercambiar entre los cromosomas padres para generar nuevos hijos. Las diversas técnicas de cruce se discuten a continuación (Kaya, 2011; Deep & Thakur, 2007; Sivanandam & Deepa, 2008; Talbi, 2009).

El operador básico es el *cruce de un punto* y su generalización el *cruzamiento de n puntos*. Estos operadores se han propuesto inicialmente para representaciones binarias, pero de igual forma se pueden utilizar para otros tipos de codificación. Si el punto de cruce se selecciona de forma adecuada, se pueden obtener hijos de buena calidad, de lo contrario puede obstaculizar

severamente la generación de buenos hijos. La Figura 2.12 muestra un operador de único punto de cruce y se puede observar que los bits próximos al punto de cruce se intercambian para generar a los hijos. Cabe señalar que la utilización de más de un punto de cruce reduce el rendimiento del AGs, ésto principalmente se origina por el hecho de que al aumentar los puntos de cruce la evolución de los cromosomas es más propensa a ser interrumpida. Sin embargo, una de las ventajas de tener más de un punto de cruce es que el espacio de búsqueda se puede recorrer de forma más exhaustiva.

Figura 2.12 Ejemplo de un operador de un punto de cruce.

El cruzamiento de un punto tiene la tendencia de cortar los cromosomas en el centro, por lo que los primeros y últimos datos de un cromosoma no se pueden pasar en conjunto a la descendencia. Si tanto la cabeza y la cola de un cromosoma contienen buena información genética, ninguno de los descendientes obtenidos directamente con un punto de cruce compartirán las dos buenas características.

Los genes que se encuentran cerca tienen más oportunidad de heredarse juntas a la próxima descendencia a través de un cruce de n puntos. Esto conduce a una correlación no deseada entre los genes próximos el uno del otro. En consecuencia, la eficiencia de un cruce de n puntos dependerá de la posición de los genes dentro del cromosoma.

El *Operador de cruce uniforme* (UPX) se basa en el uso de una malla de cruce binaria del mismo tamaño de los cromosomas, la cual indica que genes se deben heredar a partir de los padres, ésta se genera de forma aleatoria para cada par de padres. La malla está formada por un conjunto de unos y ceros, donde exista un uno el gen se copia desde el primer padre, y donde exista un cero el gen se copia desde el segundo padre, como se muestra en la Figura 2.13. El segundo descendiente se genera intercambiando los valores de la mascara de cruce

(los unos pasan a ser cero, y viceversa) o por otro lado se pueden intercambiar los padres. El número adecuado de puntos de cruce no es fija, pero tendrá un promedio de $\frac{L}{2}$ (donde L es la longitud del cromosoma).

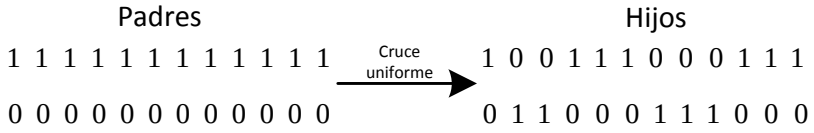


Figura 2.13 Ejemplo de un operador de cruce uniforme.

Tres padres de cruce, en esta técnica se seleccionan tres padres al azar. Se compara cada posición de los tres padres, si un gen se repite dos o tres veces se selecciona para la descendencia. Este método se muestra en la Figura 2.14. Éste operador tiene un buen funcionamiento para una representación binaria.

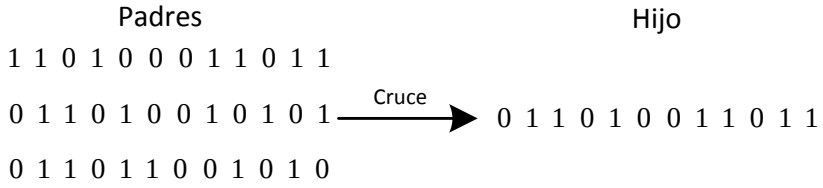


Figura 2.14 Ejemplo de un cruce de tres padres.

El *cruce aritmético* fue inicialmente propuesto por (Michalewicz, 1994), en este método las descendencias se obtienen a partir de una combinación lineal de sus padres, es decir que surgen de una función lineal que está conformada por los progenitores y por un parámetro que puede ser constante (cruzamiento aritmético estacionario) o variable (cruzamiento aritmético no estacionario). Dados los padres x_1 y x_2, el cruce aritmético crea la descendencia utilizando la ecuación 2.8.

$$x_1^{(i+1)} = \alpha x_1^{(i)} + (1-\alpha)x_2^{(i)} \qquad \text{y} \qquad x_2^{(i+1)} = \alpha x_2^{(i)} + (1-\alpha)x_1^{(i)} \tag{2.8}$$

El operador *simulated binary crossover* (SBX) intenta simular el comportamiento del operador de un punto de cruce (especialmente diseñado para una representación binaria) en una representación continua como la de números reales. El objetivo es mantener a lo largo de la descendencia los esquemas (son el conjunto de cromosomas que comparten un patrón, para mayor información consultar Alba (1999)) comunes entre los padres. Para su implementación se deben seguir los siguientes pasos:

1. Generar un número aleatorio u entre 0 y 1

2. Luego calcular el valor de $\overline{\beta}$ de la siguiente forma 2.9:

$$\overline{\beta} = \begin{cases} (2u)^{\frac{1}{n_c+1}} & \text{si } u \leq 0.5 \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{n_c+1}} & \text{de lo contrario} \end{cases} \tag{2.9}$$

   Donde, n_c es el índice de distribución y puede tomar los valores 1 ó 2

3. Por último se generan los hijos utilizando la siguiente ecuación 2.10:

$$\begin{aligned} x_1^{i+1} &= 0.5\left[\left(1+\overline{\beta}\right)x_1^i + \left(1-\overline{\beta}\right)x_2^i\right] \\ x_2^{i+1} &= 0.5\left[\left(1+\overline{\beta}\right)x_2^i + \left(1-\overline{\beta}\right)x_1^i\right] \end{aligned} \tag{2.10}$$

El *cruce heurístico* (HX en ingles heuristic crossover) fue propuesto por Wright (1991), el cual define que desde un par de padres $x^{(1)} = (x_1^{(1)} + x_2^{(1)}, \ldots, x_n^{(1)})$ y $x^{(2)} = (x_1^{(2)} + x_2^{(2)}, \ldots, x_n^{(2)})$ genera una descendencia $y = (y_1, y_2, \ldots, y_n)$, que es generada por la formula 2.11.

$$y = u(x_i^{(2)} - x_i^{(1)}) + x_i^{(2)} \tag{2.11}$$

Donde u es un número aleatorio distribuido uniformemente en el intervalo $[0,1]$ y el padre $x^{(2)}$ debe cumplir la condición de que su valor de aptitud es mejor que el padre $x^{(1)}$. Si la descendencia generada se encuentra fuera de la región factible, un nuevo número aleatorio u se genera para producir una nueva descendencia. Éste operador cuenta con algunas características interesantes que lo hacen diferente de otros operadores utilizados en una codificación real, estas son:

1. HX produce como máximo una descendencia por un par de padres
2. Hace uso de valores obtenidos por los padres desde la función de evaluación, para la generación de descendencia.
3. Utiliza la función de evaluación para direccionar la búsqueda.

En los últimos años se siguen proponiendo nuevos operadores de cruce ya sea para una codificación binaria o real, y se hacen investigaciones con el fin de comparar los operadores, y así poder concluir cuales dan los mejores resultados para un determinado problema.

Existen investigaciones que crean operadores a partir de algunas distribuciones de probabilidad, como la distribución de Laplace, llamado Laplace Crossover "LX", el cual es un operador centrado en los padres. En Deep & Thakur (2007) se realiza una comparación entre el operador propuesto LX y el operador HX concluyendo que el primero presenta mejores resultados. Además existen otras investigaciones como en Thakur, Meghwani & Jalota (2014), en el cual proponen unas variantes al operador LX, que utilizan la representación de distribución doble exponencial.

Existe la posibilidad de mezclar distintos tipos de operadores de cruce.Kaya (2011) selecciona siete operadores de cruce utilizados en la literatura (un punto, dos puntos, cruce uniforme, etc.), con el fin de generar dos nuevos operadores que ejecutarán los siete operadores de forma secuencial y aleatoria sobre las parejas de cromosomas de la misma generación. Obteniendo resultados favorables para los dos operadores propuestos, pero obteniendo mejores resultados con la ejecución aleatoria, obteniendo una mejora de 6% en el valor de aptitud comparado con el más cercano.

Por ultimo existen operadores especiales para determinados problemas, un ejemplo es el operador de cruzamiento parcialmente especificado (PMX por sus siglas en ingles), se utiliza especialmente en problemas del Agente Viajero (en inglés TSP) (Goldberg & robert, 1985). Finalmente dependiendo del tipo de problema y representación un operador puede tener un mejor comportamiento que otro.

2.3.3.7 Operador de Mutación

El operador de mutación es el último operador y se caracteriza por ser un operador unario que actúa sobre un solo individuo y que tiene por objetivo cambiar de forma arbitraria alguno de sus genes con el fin de introducir diversidad en la población y con esto permitir la exploración de todo el espacio de búsqueda, y además de impedir el quedar atrapado en mínimos locales. La probabilidad de mutación (PM) define la probabilidad de mutar cada elemento (gen) del cromosoma. Si no existe mutación, la descendencia se generara inmediatamente después del cruce, sin ningún cambio. Si por el contrario se selecciona una probabilidad de mutación de 100%, se mutaran todos los genes del cromosoma. En general, se recomiendan valores pequeños para la probabilidad ($PM \in [0.001, 0.01]$), otro método para definir éste valor es utilizar $\frac{1}{k}$, donde k es el número de variables de decisión. Existe la posibilidad de utilizar una alta probabilidad de mutación al inicio de la búsqueda e ir disminuyendo exponencialmente dicha probabilidad, aunque lo más frecuente es utilizar una probabilidad constante durante toda la búsqueda.

Algunos puntos importantes que se deben tener en cuenta en el diseño o el uso de un operador de mutación son los siguientes (Talbi, 2009):

- Debe tener la capacidad de producir soluciones válidas. Esto no siempre es posible para problemas con restricciones.
- Para llevar a cabo una búsqueda significativa, se debe contar con una localidad fuerte (pequeños cambios en genotipo, genera pequeños cambios en el fenotipo). Por el contrario si se cuenta con una localidad débil (pequeños cambios en el genotipo, provoca grandes cambios en el fenotipo), la búsqueda será aleatoria.

De acuerdo al tipo de representación pueden existir diferentes operadores de mutación, en Sivanandam & Deepa (2008) y Talbi (2009) se describen algunos. Para el caso de codificación binaria, los métodos más utilizados son la mutación Flipping, Interchanging y Reversing.

- *Voltear* (en inglés Flipping), es un método que utiliza una malla de mutación, donde exista un uno se provocara una mutación en el cromosoma seleccionado (cero a uno

y de uno a cero). En la Figura 2.15(a) se puede apreciar un ejemplo, en el cual se provocan tres mutaciones en el cromosoma padre, generando una descendencia.

- *Intercambio* (en inglés Interchanging o Swap), del cromosoma padre se seleccionan dos posiciones de forma aleatoria y los alelos correspondientes a esa posiciones se intercambian, esto se puede apreciar en la Figura 2.15(b).
- *Inversión* (en inglés Reversing), se selecciona una posición al azar y los bit próximos a esa posición se invierten y se produce el cromosoma hijo, esto se puede observar en la Figura 2.15(c).

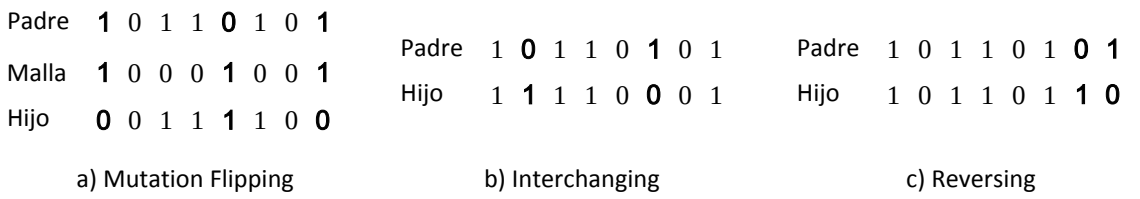


Figura 2.15 Operadores de mutación para codificación binaria.

Para una representación con valores reales, se pueden utilizar los mismos operadores propuestos para una representación binario. Pero la clase de operador más utilizada tiene la siguiente forma $x' = x + M$, donde M es una variable aleatoria. El valor de M se puede encontrar a través de los siguientes métodos: mutación aleatoria uniforme, mutación distribuida normal, mutación polinómica, operadores con distribución de Cauchy y de Laplace. Para mayor información de estos métodos revisar (Talbi, 2009).

2.3.3.8 Elitismo

Es común utilizar el operador de elitismo, el cual consiste en seleccionar al mejor o los mejores individuos de la población actual que pasaran a la siguiente generación sin sufrir ningún cambio con los operadores de cruce o mutación. Esto es útil para no perder una buena solución del problema, ya que si se utilizará una estrategia de selección por ruleta podría suceder que el mejor individuo de la siguiente generación sea peor que el mejor individuo que ya se tenía, lo cual no parece una buena opción (González, 2011).

2.3.3.9 Criterio de Convergencia

Existen diferentes criterios de paradas, a continuación se presentan algunos de ellos:

- *Número máximo de generaciones*, el algoritmo se detiene cuando el total de generaciones ejecutadas es igual al número de generaciones especificadas antes de la ejecución.
- *Lapso de tiempo*, el proceso genético debe finalizar cuando el tiempo definido se ha completado.
- *Fitness sin cambios*, el proceso genético debe finalizar si no existen cambios en el mejor fitness de la población para un número especificado de generaciones.

2.4 Algoritmo Genético Paralelo

Los Algoritmos genéticos paralelos (AGPs) no son muy diferentes de los AGs, pero se diferencian en el manejo de la población. Mientras los AGs presentan desventajas en el manejo de grandes poblaciones, obteniendo tiempos de cómputo altos. Los AGPs permiten superar éste inconveniente, gracias a la posibilidad de trabajar con múltiples poblaciones, ya sea de forma independiente o comunicadas. Las principales mejoras que presentan los AGPs son descritas en Alba (1999), entre las cuales se pueden nombrar:

1. Posibilidad de utilizar poblaciones de gran tamaño.
2. Permite disminuir el tiempo necesario en encontrar una solución.
3. Permiten buscar en diferentes sub-espacios de forma simultánea, por lo cual es menos probable que quede atrapado en regiones sub-óptimas.
4. Encuentran iguales o mejores soluciones que los AGs.

En el transcurso de toda la vida, las especies han evolucionado de forma paralela en poblaciones independientes, realizando de algún modo un intercambio genético entre ellas (más aún en este mundo globalizado). Gracias a ésta característica, se puede aumentar la cantidad de focos de búsqueda dentro del espacio de soluciones, lo cual permite tener una

mayor diversidad y por ende minimizar la probabilidad de convergencia prematura.

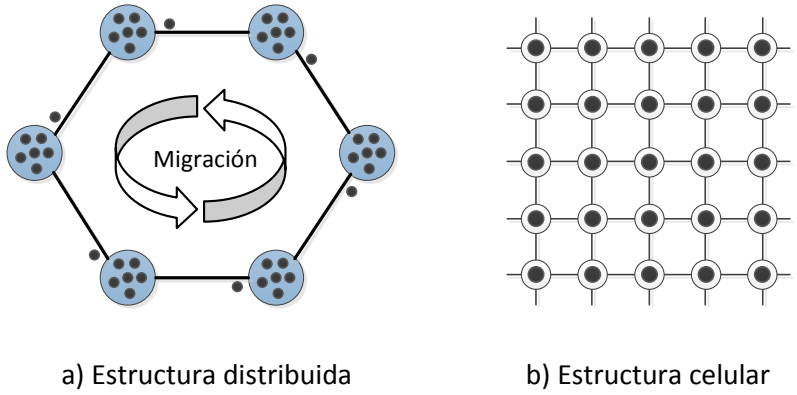


Figura 2.16 Estructura distribuida y celular de un AGPs, según (Luque & Alba, 2011).

Existen muchos tipos de algoritmos genéticos, los principales corresponden a poblaciones estructuradas de forma distribuida y celular (ver Figura 2.16). Cada uno de éstos se puede implementar en los actuales hardware como: procesadores multi-nucleo, procesadores gráficos (GPU) o FPGAs (en inglés Field Programmable Gate Array) y clúster.

2.4.1 Taxonomía

Luque & Alba (2011); Nowostawski & Poli (1999) describen los principales modelos conceptuales de AGPs que se implementan en la literatura. Estos se pueden clasificar (ver Figura 2.17) como: modelos independiente, maestro-esclavo, modelo distribuido, modelo celular y otros muy variados.

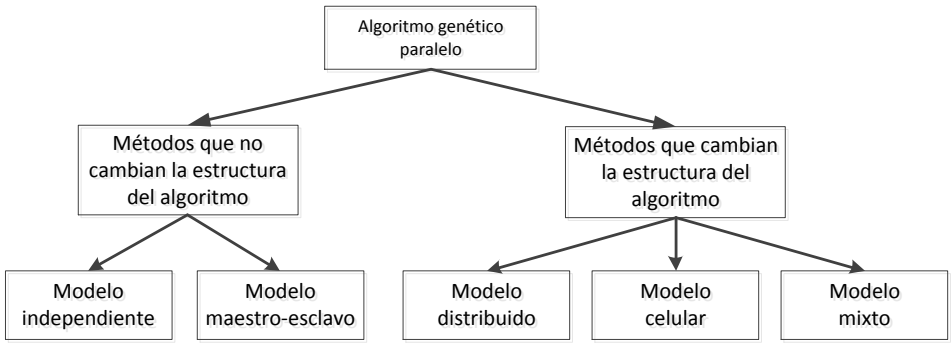


Figura 2.17 Taxonomía de los algoritmos genéticos paralelos.

2.4.2 Modelos Independientes

En los modelos independientes se utiliza un conjunto de procesadores con el fin de acelerar la ejecución de un AGs, para cada procesador se ejecuta el mismo algoritmo. En este caso no hay interacción entre las ejecuciones. Por ejemplo, se puede utilizar para ejecutar varias versiones de un mismo problema con diferentes condiciones iniciales, lo que permite adquirir mucha información del problema en un corto periodo de tiempo (ver Figura 2.18).

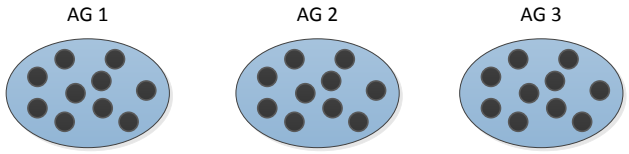


Figura 2.18 Ejecuciones independientes.

2.4.3 Modelo Maestro/Esclavo

El modelo Maestro/Esclavo distribuye la evaluación de los individuos y en algunas ocasiones la ejecución de los operadores genéticos, entre varios procesadores esclavos mientras que el bucle principal se ejecuta en un procesador maestro, con tal de que cada individuo compita con toda la población. En la Figura 2.19 se puede apreciar su esquema, el maestro envía los

parámetros necesarios para que sean calculados por los esclavos.

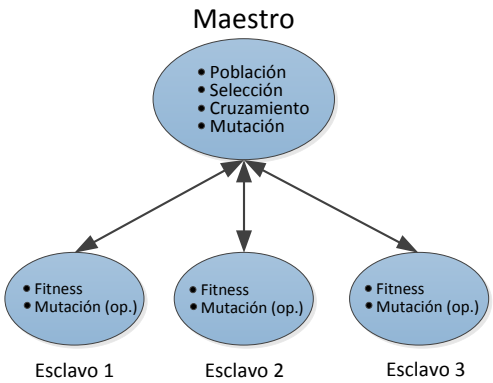


Figura 2.19 Modelo paralelo maestro-esclavo.

Los parámetros que generalmente se paralelizan son la evaluación de la aptitud y la mutación, ya que para su ejecución solo se necesita la información de solo un individuo. Al no necesitarse toda la población durante esta fase no es necesaria una comunicación entre los procesadores. Este modelo es fácil de implementar y la exploración del espacio de búsqueda es conceptualmente idéntica a la ejecución de un AGs, la única diferencia es el tiempo de ejecución.

Este modelo es eficiente cuando la función de evaluación es muy costosa de resolver, ya que la sobrecarga de comunicación entre el maestro y los esclavos es insignificante con respecto al tiempo de evaluación de la aptitud. Sin embargo, tiende a presentar cuellos de botellas cuando se utilizan muchos procesadores.

2.4.4 Modelo Distribuido

Las poblaciones están estructuradas en pequeñas sub-poblaciones aisladas unas de otras. La característica principal de este tipo de algoritmo es que utiliza un operador genético adicional de migración, mediante el cual, una copia de un individuo de una sub-población particular emigra hacia otra, como una forma de compartir material genético entre las mismas. Éste

modelo se muestra en la Figura 2.16(a).

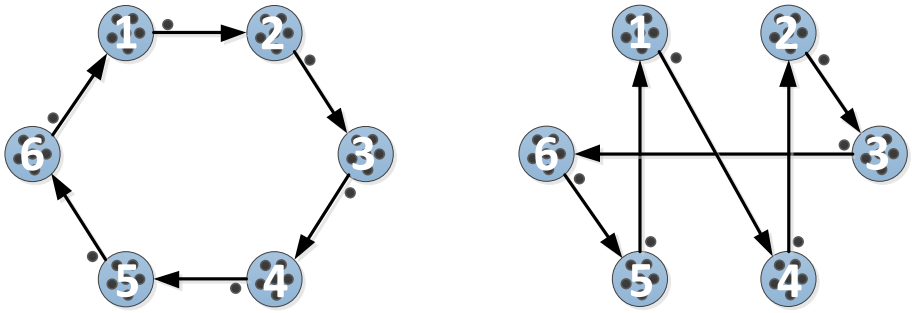
Ésta estrategia de paralelización también es conocida en la literatura como modelo de islas y algoritmo genético de grano grueso.

Los operadores genéticos se ejecutan dentro de cada isla, lo cual permite que cada isla pueda explorar el espacio de búsqueda en diferentes regiones. Además cada isla puede tener parámetros con diferentes valores, generalmente este tipo de algoritmo son llamados AGs heterogéneos.

Es necesario definir los principales parámetros de migración que influyen en la eficiencia del algoritmo, a continuación se describen de forma breve:

- *Migración Gap*, define cada cuanto se deben hacer los intercambios de individuos entre las sub-poblaciones.
- *Tasa de migración*, determina el número de individuos que serán intercambiados.
- *Selección/Reemplazo de migración*, define como serán seleccionado los emigrantes, y como se reemplazarán los inmigrantes.
- *Topología*, define los vecinos de cada isla. Se pueden agrupar en modelos de anillo y aleatorio (ver Figura 2.20).

Es recomendable migrar proporciones pequeñas de individuos (en el rango del 20%), con el objetivo de minimizar el uso del ancho de banda entre procesadores. De igual forma que el AGs se pueden utilizar técnicas de selección como elitismo y selección por torneo.



a) Migración en anillo b) Migración aleatoria

Figura 2.20 Modelos de migración.

2.4.5 Modelo Celular

El modelo celular consta normalmente de una sola población conceptual, donde cada unidad de proceso cuenta con unos pocos individuos (por lo general uno o dos), esta es la razón de que sea conocido como de grano fino. La principal característica de este modelo es que está estructurado en barrios, en el que los individuos sólo pueden interactuar con sus vecinos. Por lo tanto, buenas soluciones se generan en diferentes áreas de la topología global, y se extiende lentamente a lo largo de toda la estructura, éste modelo se puede apreciar en la Figura 2.16(b).

Inicialmente fueron diseñados para trabajar en máquinas masivamente paralelas, pero con la pérdida de popularidad que ha sufrido este tipo de computadores, se están utilizando en unidades de procesamiento gráfico (GPU) y FPGAs.

2.4.6 Modelos Híbridos

Es común encontrar implementaciones de difícil clasificación. En general, se les llama algoritmos paralelos híbridos ya que implementan características de los modelos anteriormente expuestos.

En la Figura 2.21 se pueden apreciar tres tipos de modelos híbridos. En el primero de ellos se aplican dos niveles de paralelización, multipoblación en el nivel superior, y un grano

fino en el nivel inferior. Otra posibilidad es combinar un algoritmo multipoblación en el nivel superior y uno maestro-esclavo en el nivel inferior. Un tercer método, es utilizar algoritmo multipoblación en el nivel superior e inferior. Por último se puede apreciar que en los tres casos el nivel más alto de paralelización es un algoritmo distribuido.

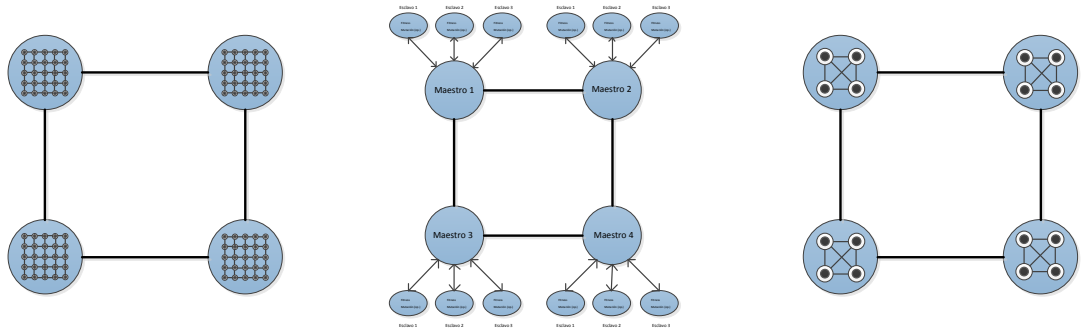


Figura 2.21 Modelos mixtos, según (Luque & Alba, 2011)

Aunque estas combinaciones pueden dar lugar a interesantes y eficaces nuevos algoritmos, tienen la necesidad de agregar nuevos parámetros para tener en cuenta la estructura implementada.

2.5 Frameworks de Optimización con Metaheurísticas

2.5.1 Tipos de Frameworks

En la literatura existen numerosos framework con los que se pueden implementar distintas metaheuristicas. Para éste trabajo las principales características que se buscan en un framework son: buen rendimiento en la implementación del algoritmo, principalmente que cuente con la posibilidad de implementar AGPs ya sea en arquitecturas de memoria compartida o no, que tenga soporte técnico y que cuente con una buena documentación.

Los frameworks para la optimización con metaheuristica (en inglés metaheuristic optimization frameworks "MOFs") se pueden definir según Parejo (2013) como, "conjunto de herramientas de software que permiten una implementación correcta y reutilizable de un conjunto de metaheurísticas, que proporcionan mecanismos para acelerar la aplicación de heurísticas subordinadas (posiblemente técnicas de codificación y operadores específicos), que son necesarias para resolver un determinado problema". Permiten desarrollar metaheurísticas de una mejor forma desde el punto de vista del costo de la implementación y el esfuerzo.

Tabla 2.1: Ventajas y desventajas en la utilización de MOFs (Parejo, 2013)

Ventajas	Desventajas
Reducción del esfuerzo en la implementación y capacidad para aplicar diversas técnicas y variantes	Curva de aprendizaje empinada
Herramientas adicionales para la resolución de problemas (monitoreo, reporte, computación paralela y distribuida)	Es necesario un conocimiento avanzado para la adaptación, falta de flexibilidad para la adaptación de metaheurísticas
Optimizado y ejecución sin errores (excepto para las extensiones y adaptación creadas por los usuarios o los errores no detectados que podrían estar presentes en el MOF)	Complejidad inducida (cuando la depuración y pruebas) y dependencias adicionales
Usuarios con poco conocimientos pueden usar el framework como un entorno de desarrollo de aplicaciones, y además como una ayuda metodológica	La adecuada elección del MOF puede ser complicada, ya que el cambio de uno a otro tiene altos costos en tiempo y esfuerzo

La principal ventaja del uso de MOFs es que proporcionan un conjunto de metaheurísticas totalmente funcionales y optimizadas. Además, proporcionan mecanismos que facilitan la implementación de heurísticas como, la representación de la población, generación de la población inicial y los distintos tipos de operadores. Sólo se debe prestar atención a los elementos que están directamente relacionados con el problema, dejando de lado aspectos que no dependen de él. En la Tabla 2.1 se describen las principales ventajas y desventajas de estas herramientas.

Tabla 2.2: Lista de frameworks de Computación Evolutiva.

Nombre	Fecha de inicio	Última actualización	Lenguaje de programación	Plataforma	Algoritmo Evolutivo	Computación paralela y distribuida
JCLEC/KEEL	2004	2010	Java	Multiplataforma	✓	✖
JGAP	2002	2013	Java	Multiplataforma	✓	✓
Watchmaker	2006	2010	Java	Multiplataforma	✓	✓
DREAM	2000	2003	Java	Multiplataforma	✓	✓
Osgiliath	-	2013	Java	Multiplataforma	✓	✓
JavaEva/EvA2	-	2013	Java	Multiplataforma	✓	✓
FOM	2003	-	Java	Multiplataforma	✓	✖
Opt4J	2007	2014	Java	Multiplataforma	✓	✖
jMetal	2008	2014	Java	Multiplataforma	✓	✓
ECJ	2003	2011	Java	Multiplataforma	✓	✓
GA Toolkit	1998	2003	Java	Multiplataforma	✓	✖
Jenes	2006	2014	Java	Multiplataforma	✓	✓
MOEA Framework	2011	2014	Java	Multiplataforma	✓	✖
PISA	2004	2010	Java	Multiplataforma	✓	✖
Jaga	2004	2008	Java	Multiplataforma	✓	✖
EasyLocal++	2001	2011	C++	Unix	✖	✖
UOF	2005	2006	C++	Unix	✓	✖
ECF	2011	2014	C++	Unix\Ubuntu	✓	✓
Mallba	2000	2007	C++	Unix	✓	✓
ParadisEO/EO/MOEO/PEO	2003	2013	C++	Multiplataforma	✓	✓
Galib-PVM	1995	2007	C++	Multiplataforma	✓	✓
Galib-MPI	-	2012	C++	Multiplataforma	✓	✓
Open BEAGLE	1999	2010	C++	Multiplataforma	✓	✓
Geneva	2008	2014	C++	Unix\Ubuntu	✓	-
GAUL	2000	2006	C/C++	Unix	✓	✓
PGAPack	1990	2009	Fortran/C	Unix	✓	✓
HeuristicLab	2002	2013	C#	Windows	✓	✓
MPIKAIA	1998	2003	Fortran	Unix	✓	✓

Antes de decidir que framework se utilizara para la implementación de este trabajo de título, se realizó una búsqueda exhaustiva, con el fin de recopilar todos los MOFs existentes en la literatura científica. Las fuentes de información que se utilizaron para la recopilación de los frameworks existentes fueron: IEEE Xplore, ISI Web of Knowledge, SpringerLink, Science Direct y Google Académico.

En la Tabla 2.2 se muestran todas las librerías o framework encontrados en las bases de datos anteriormente nombradas. En ella se dan a conocer las siguientes características: Nombre, fecha de inicio, última actualización, plataforma, si cuenta con implementación paralela o distribuida, lenguaje de programación.

En la literatura existen algunos trabajos, en los que se hacen comparaciones entre los distintos MOFs, pero solo hacen el estudio de forma general sin enfocarse en una determinada metaheurísitica Parejo, Ruiz-Cortés, Lozano & Fernandez (2012); Gagné & Parizeau (2006). Como para este trabajo se necesita un MOFs que cuente con herramientas para la implementación de un AGPs.

Según el trabajo de Parejo et al. (2012), si se quieren implementar algoritmos evolutivos en java los MOFs ECJ, JCLEC y EvA2 son altamente competitivo. Por otro lado sí se quiere utilizar el lenguaje de programación C++ como base, los que tienen mejores características son ParadisEO y MALLBA.

Los criterios de filtrado utilizados para descartar los MOFs que no se ajustan a este trabajo son:

1. El MOFs debe estar implementado en un lenguaje de programación orientado a objeto preferentemente C++. A partir de éste requisito se eliminaron un total de 18 MOFs.
2. El MOFs debe estar activo. Se considerara como proyecto abandonado, si no cuenta con nuevas versiones o artículos publicados en los últimos 5 años. Con este criterio se eliminaron cuatro MOFs.
3. Debe tener la capacidad de implementar un algoritmo genético en su forma paralela. Este criterio elimino dos framework.
4. De preferencia que sea multiplataforma.
5. Y por último se utilizan las sugerencias propuestas por los trabajos de Luque & Alba (2011) y Parejo et al. (2012) como último filtro. En la Tabla 2.3 se pueden apreciar los MOFs que sobreviven a los criterios expuestos.

De acuerdo con el trabajo realizado por (Parejo, 2013), los framework mejor catalogadas son:

Tabla 2.3: MOFs sobrevivientes.

Nombre	Lenguaje de programación	Plataforma
ParadisEO/EO/MOEO/PEO	C++	Multiplataforma
Galib-MPI	C++	Multiplataforma
Open BEAGLE	C++	Multiplataforma

- ECJ el cual cumple con la mayor cantidad criterios propuestos por el trabajo, entre los que se puede nombrar: cuenta con una gran biblioteca de metaheurísticas, incluidos los algoritmos evolutivos, es posible ejecutarlo en los principales sistemas operativos (Windows, Linux y Mac). En este trabajo este framework quedo descartado por el hecho de que no cumple con las necesidades, principalmente por el motivo de que está desarrollado en lenguaje Java.

- En segundo lugar quedó el framework Paradiseo el cual se destaca por contar con una gran variedad de metareurísticas para implementar y cuenta con módulos para el desarrollo de algoritmos evolutivos, además se destaca en la implementación de metaheurísticas hibridas y paralelas.

Estos sistemas están dotados de estructuras de programación "generales" destinados a facilitar la implementación de cualquier modelo de AGPs, el usuario es quien debe particularizar estas estructuras generales para definir su propio algoritmo.

Finalmente se selecciona el framework ParadisEO por contar con la opción de implementar muchas variantes para el paralelismo y por ser tan completo como MALLBA o ECJ con respecto a los modelos implementados y, además por contar con una comunidad activa hasta la fecha.

2.5.2 Framework ParadisEO

ParadisEO es un framework desarrollado inicialmente por el instituto nacional de investigación en informática y automática de Francia. Tiene por objetivo reutilizar metaheurísticas

híbridas, metaheurísticas multi-objetivo y metaheurísticas paralelas y distribuidas. Entre las metaheurísticas propuestas se pueden encontrar: algoritmos evolutivos, búsqueda local, optimización por enjambre partícula, métodos multi-objetivo, etc.

Para la implementación de las distintas técnicas, ParadisEO cuenta con cuatro módulos: Paradiseo-EO (metaheurísticas basadas en población), Paradiseo-MOEO (metaheurísticas multi-objetivo), Paradiseo-MO (metaheurísticas de una solución), Paradiseo-PEO-SMP (metaheurísticas hibridas, paralelas y distribuidas).

El módulo Paradiseo-EO, cuenta con un conjunto de herramientas que pueden ser usadas de forma independiente de la metaheurística, la mayoría están escritas usando template lo que permite escribir código de forma genérica. La implementación de un algoritmo evolutivo consta de las siguientes etapas: una representación, inicialización de la población, función de evaluación, operadores de variación y motor evolutivo, en éste último se encuentran los operadores de selección y reemplazo. Generalmente solo es necesario programar la función de evaluación, aunque existe la posibilidad de desarrollar operadores de variación propios, derivándolos de las correspondientes sub-clases. Además se pueden utilizar tres tipos de genotipos; binario, real y permutación.

El módulo Paradiseo-PEO-SMP, de igual forma que el módulo EO hace uso de los template, y además puede utilizar las herramientas presentes en los demás módulos. Algunos de los problemas que se pueden resolver con éste módulo se describen a continuación:

- Si se requiere implementar tres algoritmos de optimización idénticos y que además cooperen entre ellos. Para éste caso se pueden implementar técnicas de migración entre las metaheurísticas, con el objetivo de incrementar la calidad de las soluciones.
- Cuando una aplicación debe ser implementada de forma distribuida. Éste tipo de problema se presenta cuando un usuario cuenta con varias máquinas y en cada una quiere implementar una determinada metaheurística.
- Cuando las metahuerísticas deben intercambiar el mismo tipo de datos.

Como se especificó en la sección 2.5 la principal ventaja de éste framework con respecto a los demás es principalmente que no está restringido solo al uso de algoritmos evolutivos, y además permite implementar distintas arquitecturas paralelas.

2.6 Computación de Alta Performance

Para la implementación del AGPs es necesario conocer los conceptos básicos de computación de alta performance (CALP), principalmente este estudio se enfocara en las arquitecturas de computación paralela de memoria compartida (SMP) y memoria distribuida (MPP), ya que son las que se pueden implementar con el framework ParadisEO. Se debe agregar que estas arquitecturas se encuentran en el grupo de arquitecturas MIMD la cual es una categoría propuesta por (Flynn, 1972). Sin embargo se describirán de forma breve el resto de arquitecturas, con el fin de tener una visión global de la taxonomía existente de computación paralela, ver Figura 2.22.

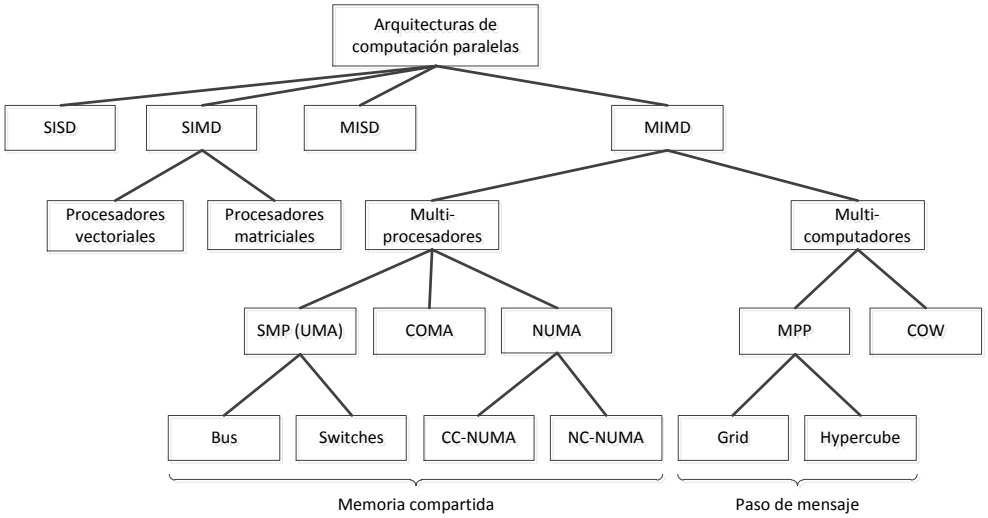


Figura 2.22 Taxonomía de computación paralela, según (Tanenbaum, 2012).

La bibliografía para esta sección se puede encontrar en cualquier libro de arquitectura de computadores, y principalmente se recomienda para mayor información la lectura de los siguientes libros (Hennessy & Patterson, 1993; Parhami, 2007; Tanenbaum, 2012), además se

utilizara como referencia un documento de computación paralela confeccionado por Rodríguez & Risso (2009).

2.6.1 Clasificación de las Arquitecturas

Flynn (1972) propuso un sencillo modelo para clasificar todas las arquitecturas de computadores, aunque este modelo no es detallado, ya que algunas máquinas pueden estar en más de una categoría. Se seguirá está taxonomía por el hecho de ser fácil de comprender, y también por ser la más utilizada en la literatura.

De acuerdo con la taxonomía propuesta, las arquitecturas se pueden clasificar de la siguiente forma:

- Flujo único de instrucciones, flujo único de datos (SISD, el uniprocesador)
- Flujo único de instrucciones, flujos múltiples de datos (SIMD)
- Flujos múltiples de instrucciones, flujo único de datos (MISD)
- Flujos múltiples de instrucciones, flujos múltiples de datos (MIMD)

2.6.1.1 Arquitectura SISD

Son los modelos convencionales de computación (modelo convencional de Von Neumann), como se muestra en la Figura 2.23. Esta arquitectura está diseñada con el fin de dar soporte al procesamiento secuencial clásico. Si bien esta arquitectura mejoro su funcionamiento gracias algunas modificaciones (arquitecturas CISC, RISC, aceleración del ciclo de reloj, etc.), estas no fueron suficiente y el ritmo de mejoramiento se desacelero, por lo que se diseñaron nuevas alternativas.

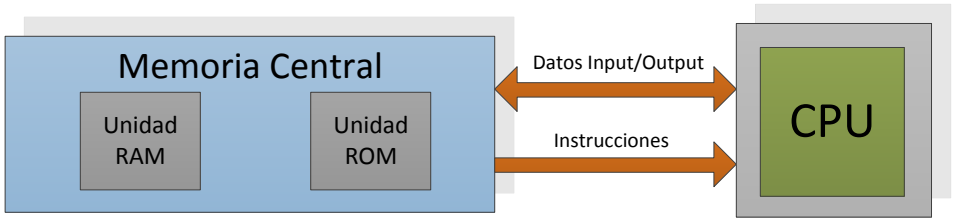


Figura 2.23 Arquitectura Von Neumann

2.6.1.2 Arquitectura MISD

Es una arquitectura donde varias CPU ejecutan distintas instrucciones para el mismo dato, en la Figura 2.24 se puede apreciar su estructura. Dispone de N procesadores, cada uno con su unidad de control que comparte una unidad de memoria. Las computadoras de esta clase no han encontrado una aplicación extensa y raramente son usadas.

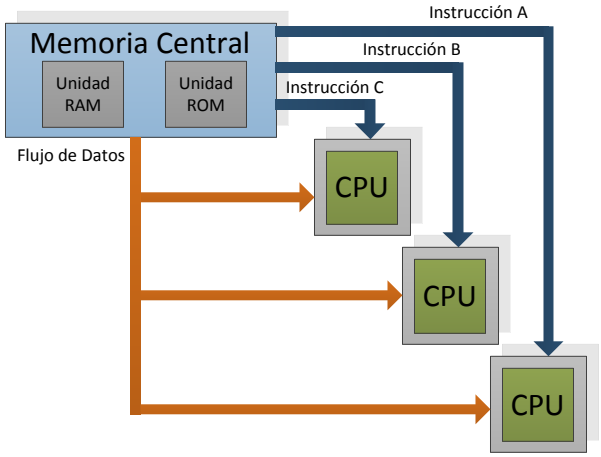


Figura 2.24 Arquitectura de Múltiples Instrucciones y Único Dato.

2.6.1.3 Arquitectura SIMD

En las arquitecturas SIMD una misma instrucción se ejecuta para un conjunto de datos de forma paralela, cuenta con un conjunto de unidades de procesamiento elementales que se encuentran bajo la supervisión de una única unidad de control. Una de sus desventajas es la

dificultad que presenta cuando se intenta hacer aritmética de alta precisión, por la simplicidad de cada celda de proceso. Algunas aplicaciones de esta arquitectura son en áreas donde se requiere operaciones aritméticas de baja precisión sobre estructuras de datos regulares como en procesamiento de señales e imágenes, gestión de base de datos y minería de datos. En la Figura 2.25 se muestra de forma general este tipo de arquitectura.

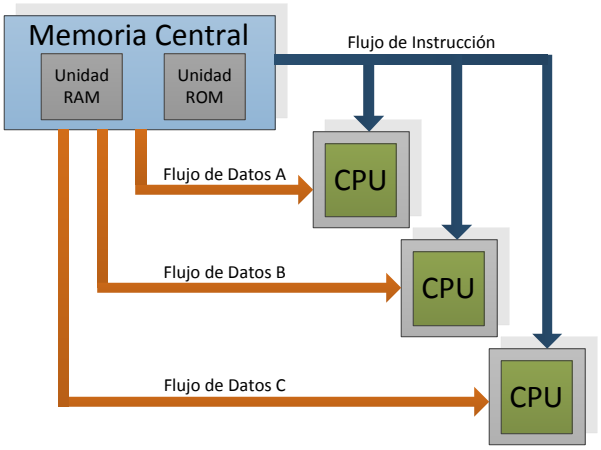


Figura 2.25 Arquitectura de Única Instrucción y Múltiples Datos.

2.6.1.4 Arquitectura MIMD

Johnson (1988) propuso una mayor clasificación para estas máquinas en base a su estructura de memoria (global o distribuida) y en el mecanismo usado para la comunicación/sincronización (variables compartidas "shared variables" o paso de mensajes "message pasing"), en la Figura 2.26 se puede apreciar las cuatro categorías propuestas.

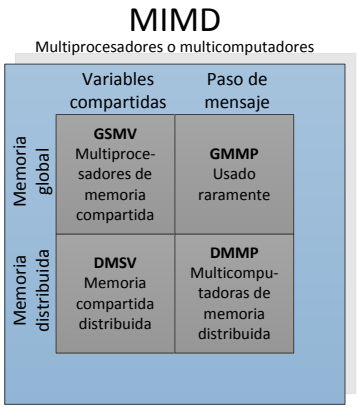


Figura 2.26 Clasificación Johnson de sistemas de cómputo, según (Parhami, 2007).

De estas categorías las más comunes son, las arquitecturas GMSV que es conocida como "multiprocesamiento (memoria compartida)" y la DMMP que se conoce como "multicomputación (memoria distribuida)".

Los *multiprocesadores de memoria compartida*, se definen como un conjunto de procesadores que comparten un área de almacenamiento. Algunas de sus ventajas son la facilidad de programación ya que es intuitiva y conceptualmente similar a la programación ordinaria, la existencia de muchas librerías y la gran potencia con la que cuentan los procesadores individuales actuales. Diferentes técnicas han sido adoptadas para asegurar la consistencia de los datos cuando más de un procesador intenta acceder a ellos de forma simultánea y para evitar el cuello de botella, una de las principales es el uso de múltiples cachés y coherencia de cachés. La Figura 2.27 muestra un sistema MIMD típico de memoria compartida.

Las *multicomputadoras de memorias holgadamente acopladas*, se forman al interconectar un conjunto de computadoras independientes mediante una red de interconexión que les permite comunicarse mediante el paso de mensajes. Permite la comunicación de datos por medio de una red ethernet, además de resolver el problema de manejo de recursos y sincronización de procesos, por ultimo una gran ventaja de esta arquitectura es la facilidad de escalamiento. Tienen como desventaja que la responsabilidad en la distribución apropiada de los datos a los procesadores y la programación en paralelo recae totalmente sobre el programador, ya que en la actualidad no existen herramientas de generación automática de código

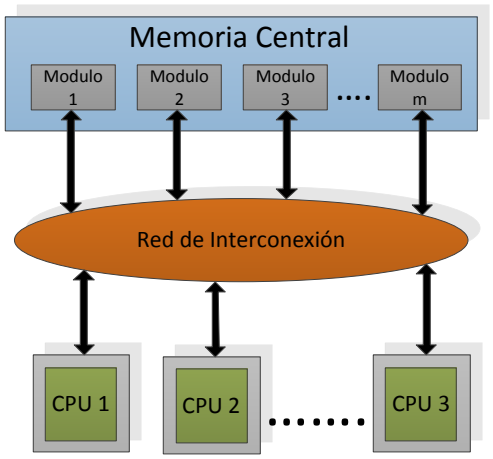


Figura 2.27 Modelo típico multiprocesador de memoria compartida.

paralelo en memoria distribuida. La Figura 2.28 muestra un sistema MIMD típico de memoria distribuida.

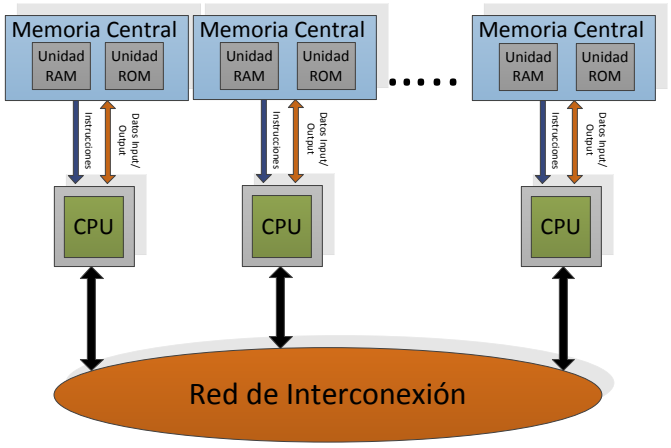


Figura 2.28 Modelo típico multicomputador de memoria distribuida.

2.6.2 Arquitecturas MIMD Modernas

En las arquitecturas modernas se destacan las arquitecturas SMP (Simetric Multiprocesing) y MPP (Massively Parallel Processing), por el hecho de que pueden ser implementadas por

el framework seleccionado en secciones anteriores.

2.6.2.1 SMP (Simmetric Multiprocessing)

Ésta arquitectura es también conocida como UMA (Uniform Memory Access), principalmente hace uso de un bus para interconectar una moderada cantidad (2-64) de procesadores (cada uno con su propia caché privada), con el fin de compartir los recursos de memoria y disco. Una de sus desventaja es que está acotado en la cantidad de procesadores que se pueden utilizar para la implementación de un procesamiento paralelo a gran escala, que implica el uso de cientos a miles procesadores, por el motivo de que habría un aumento en la competencia por el uso del bus. Para una configuración de dos o tres CPUs, la competencia por el bus es manejable, con más de 64 es prácticamente inviable. Finalmente el sistema se encuentra limitado al ancho de banda del bus. Actualmente para disminuir el uso del bus cada procesador cuenta con su propia memoria cache, pero a partir de ésta se genera un nuevo fenómeno que disminuye el rendimiento del sistema que es conocido como coherencia de caché. Cada vez que se realiza una escritura en memoria, si una CPU posee en caché un cierto valor que a su vez corresponde a un valor almacenado en la memoria compartida, y otra CPU escribe en esa dirección, es necesario informar que dicha escritura se realizó. Como es evidente, a medida que aumenta el número de CPUs, informar este evento resulta más caro.

Para mejorar éste tipo de arquitectura se deben implementar nuevas formas de interconexión entre las CPUs, como emplear el uso de conmutadores tipo *crossbar* y el uso de redes de conmutación *multietapas*.

En este tipo de arquitectura como todos los procesadores tienen igual acceso a todos los dispositivos periféricos el sistema es denominado Simetric Multiprocessing. En la Figura 2.29 se aprecia una arquitectura de multiprocesador simétrico típico.

2.6.2.2 MPP (Massively Parallel Processors)

Ésta arquitectura se caracteriza por tener un enfoque “shared nothing”, lo que significa que las unidades de procesos tienen asociado sus propios recursos de memoria, I/O y procesadores. La comunicación entre los nodos se realiza mediante el uso de la técnica de paso de mensajes,

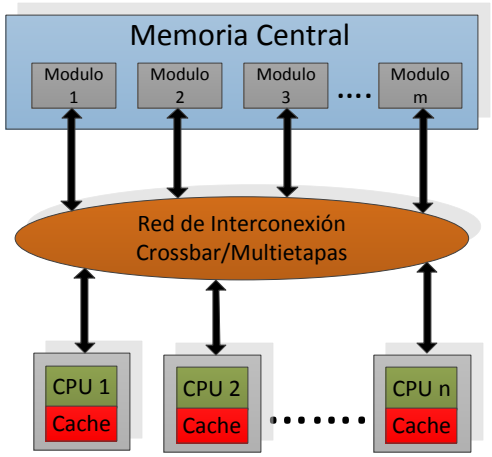


Figura 2.29 Arquitectura de un multiprocesador simétrico con memoria caché.

utilizando líneas de alta velocidad que pueden ser seriales o redes Ethernet.

Algunas de las principales ventajas de implementar este tipo de arquitectura son: la facilidad de escalabilidad y la proporcionalidad del rendimiento de acuerdo a la cantidad de nodos conectados (más nodos conectados mayor rendimiento). Por otro lado cuenta con la desventaja de ser una arquitectura muy costosa si se implementa desde cero y de ser muy compleja de programar.

2.6.3 Métricas de Desempeño

Un algoritmo paralelo depende del número de procesadores y de la arquitectura paralela en donde se implemente. Algunos de los indicadores de rendimiento más antiguos buscan evaluar la escalabilidad de los algoritmos paralelos en el aumento de la velocidad y su eficiencia. La escalabilidad de un algoritmo paralelo es el rendimiento obtenido a partir de un cierto número de procesadores.

Si se requiere medir el rendimiento relativo (el rendimiento absoluto se utiliza cuando se conoce el tiempo absoluto en encontrar una solución óptima) entre un algoritmo secuencial y uno paralelo, se puede utilizar el speedup S_N (da el incremento de velocidad utilizando

una arquitectura multiprocesador), el cual está definido por la Formula 2.12, donde T_1 es el tiempo que demora un algoritmo en ejecutarse sobre una arquitectura de un procesador y T_N es el tiempo que demora el mismo algoritmo sobre una arquitectura de N procesadores.

$$S_N = \frac{T_1}{T_N} \qquad \text{donde:} \qquad \begin{aligned} Desaceleración &\Leftrightarrow S_N < 1 \\ Speedup\,lineal &\Leftrightarrow S_N = N \\ Speedup\,sublineal &\Leftrightarrow S_N < N \\ Speedup\,superlineal &\Leftrightarrow S_N > N \end{aligned} \tag{2.12}$$

Una ganancia sublineal se presenta debido a la sobrecarga en la comunicación y sincronización. El caso de desaceleración significa que el tiempo de un algoritmo secuencial es más pequeño que el tiempo de un algoritmo paralelo, éste es el peor caso que se puede presentar. En la Figura 2.30 se presentan gráficamente todas las posibilidades.

Para utilizar el concepto de speedup en el campo de los algoritmos evolutivos paralelos se requieren algunas consideraciones especiales sobre el criterio de parada, ya que se pueden implementar dos métodos diferentes para la comparación de los algoritmos.

- Número fijo de iteraciones: Esta condición es la más usada para evaluar la eficiencia de una metaheurística paralela. Se enfoca en comparar los tiempos de ejecución entre los algoritmos sin prestar atención a la calidad de la solución.

- Convergencia a una solución predefinida: Esta definición es una de las más interesantes para evaluar la eficiencia de un algoritmo paralelo. Sólo es válido para comparar el algoritmo paralelo con su versión secuencial, o comparar la ejecución del mismo algoritmo paralelo sobre diferentes plataformas con m procesadores.

Según Alba (1999), para comparar el speedup entre algoritmos es recomendado encontrar soluciones con calidad equivalente entre los algoritmos ejecutados, ya sean secuenciales o paralelos. Detener la ejecución de los algoritmos utilizando como criterio de parada la cantidad de iteraciones no es un criterio justo ni significativo para la comparación. Además en Talbi (2009) para una comparación del rendimiento más justa se sugiere utilizar la misma

semilla para la generación de números aleatorios.

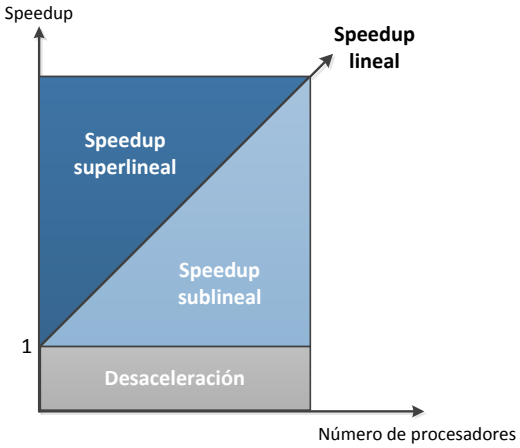


Figura 2.30 Speedup de un programa paralelo, según (Talbi, 2009).

Para los algoritmos evolutivos que utilizan como condición de parada la calidad de la solución, no se puede utilizar directamente la Formula 2.12. La definición original se puede extender a la media del speedup (Formula 2.13), conocida como speedup debil:

$$S_N = \frac{E[T_1]}{E[T_N]} \tag{2.13}$$

Otro de los parámetros a considerar es la eficiencia E_N, la cual está representada como el speedup S_N dividido por el número de procesadores N (ver Formula 2.14).

$$E_N = \frac{S_N}{N} \tag{2.14}$$

La eficiencia se define como el tiempo de utilización de los procesadores presentes para la ejecución de un programa, o en otras palabras, que tan bien un algoritmo utiliza los procesadores extras. Una eficiencia del 100% significa que los procesadores están siendo utilizados por completo y durante todo el tiempo de ejecución del algoritmo.

CAPÍTULO 3

Formulación del problema

En este capítulo se presenta una caracterización formal del problema de localización de nodos, se describe una formulación general que puede utilizarse como base para la solución del problema. Además, se describen algunas de las restricciones consideradas para el desarrollo del modelo.

3.1 Formulación

Una WSN está compuesta de n nodos. Cada nodo cuenta con un rango de comunicación r, que son distribuidos en un área cuadrada bi-dimensional Q. Para simplicidad del problema, los enlaces de comunicación se consideran simétricos, es decir, para cualquier par de nodos u y v, u se encuentra en el rango de v si y sólo si, v alcanza a u con la misma fuerza de la señal w. Por lo tanto, la red se puede representar por un plano Euclidiano con las siguientes características (Boukerche, Oliveira, Nakamura & Loureiro, 2007):

- $V = v_1, v_2, \ldots, v_n$ conjunto de nodos sensores.
- $\langle i, j \rangle \in E$ (donde E es el conjunto de enlaces entre los nodos vecinos) si y sólo si v_i cubre v_j, es decir, la distancia entre v_i y v_j es menor que r.
- $w(e) \leq r$ es el peso del borde $e = \langle i, j \rangle$, que es la distancia entre v_i y v_j.

En un plano Euclidiano, cada nodo cuenta con una coordenada $(x_i, y_i) \in \Re^2$ en un espacio bi-dimensional, el cual representa la ubicación del nodo i dentro de Q.

Para la formulación del problema de localización de WSN es necesario conocer los principales elementos y, junto con ello, las características propias de cada elemento y del conjunto.

- *Nodos Desconocidos,* υ. También pueden ser conocidos como nodos libres, son nodos de la red que no conocen su posición. Permitir a éstos nodos encontrar su posición es el principal objetivo de los algoritmos de localización.
- *Nodos Encontrados,* ς. Estos nodos inicialmente son desconocidos, pero utilizando alguna técnica de localización se puede obtener su posición relativa. La cantidad de nodos encontrados y el error estimado de posición son los principales parámetros de calidad de un sistema de localización.
- *Nodos anclas,* β. También conocidos como landmarks o balizas, son nodos que no necesitan ser localizados, ya que sus posiciones son obtenidas de forma manual o utilizando algún hardware externo como GPS. Estos nodos son una pieza fundamental para los algoritmos de localización.

El problema de localización puede ser simplificado y definido de la siguiente forma:

Dado una red multi-salto $G = (V, E)$, y un conjunto de nodos anclas β y sus respectivas posiciones (x_b, y_b), para todo $b \in \beta$, se quiere encontrar la posición x_u de todos los nodos desconocidos $u \in \upsilon$, transformando a los nodos desconocidos en nodos encontrados ς.

3.1.1 Problema de optimización

Dado un conjunto V de n nodos v_i, $1 \leq i \leq n$, un conjunto de nodos anclas β_j, $1 \leq j \leq K < n$ (K cantidad de nodos anclas), y un conjunto de distancias estimadas $\delta_{i,j}$, con $1 \leq i,\ j \leq n$ y $i \neq j$, se debe determinar la localización de cada nodo υ_l, $K < l \leq N$, de tal manera que el error de localización sea mínimo (Molina & Alba, 2011).

Sean a y b dos puntos en el espacio Euclidiano con coordenadas (x_a, y_a) y (x_b, y_b) respectivamente, (x'_a, y'_a) y (x'_b, y'_b) sus coordenadas estimadas (obtenidas a través de alguna técnica de estimación, sección 2.2.3), se define lo siguiente:

- Distancia real $d_{a,b} = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$
- Distancia medida o estimada $\delta_{a,b}$ (obtenida por algún método de la sección 2.2.2)
- Distancia calculada $c_{a,b} = \sqrt{(x'_a - x'_b)^2 + (y'_a - y'_b)^2}$
- Error de distancia medida $\varepsilon_{a,b} = \delta_{a,b} - d_{a,b}$
- Error de distancia calculada $\varepsilon'_{a,b} = c_{a,b} - d_{a,b}$
- Error de localización $\varepsilon_a = \sqrt{(x_a - x'_a)^2 + (y_a - y'_a)^2}$, $\varepsilon_b = \sqrt{(x_b - x'_b)^2 + (y_b - y'_b)^2}$

Finalmente, el objetivo de la localización es minimizar el error de localización ε_i para todos los nodos n_i, pero para las definiciones anteriores es necesario contar con los valores de x_i e y_i, los cuales en un escenario real son desconocidos (además son los valores que se intentan encontrar), por lo tanto, para resolver este inconveniente se debe utilizar una función de fitness para la optimización del problema.

La función de guiado seleccionada en la sección 2.2.3.3, es la norma del error. Para ésta función se utiliza el error de distancia media, el cual se define como $\varepsilon_{i,j} = c_{i,j} - \delta_{i,j}$, luego la función de optimización para todos los nodos dentro de una red se representa con la ecuación 3.1, la cual describe que tan buenas son las coordenadas estimadas, mientras menor sea el error mejores serán las coordenadas estimadas.

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n-k} |\varepsilon_{i,j}| \qquad \text{y} \qquad \Delta := \begin{pmatrix} \delta_{1,1} & \delta_{1,2} & \cdots & \delta_{1,n} \\ \delta_{2,1} & \delta_{2,2} & \cdots & \delta_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{n,1} & \delta_{n,2} & \cdots & \delta_{n,n} \end{pmatrix} \tag{3.1}$$

3.1.2 Restricciones del problema

Una de las restricciones básica para este problema corresponde a la región de trabajo. Todos los nodos deben encontrarse dentro de una región permitida (a,b), donde $a \leq x_i$ y $b \geq y_i$. Aunque es una restricción básica es de gran importancia para el resto del trabajo.

Cuando un nodo sólo tiene conectado dos nodos, existen dos posibles posiciones para el nodo desconocido. Si se tiene en cuenta que una de las posiciones se encuentra próxima a un nodo al que no debería tener ninguna aproximación, debido a que no existe comunicación entre ellos se podría descartar. En la Figura 4.1 se presenta un ejemplo de éste inconveniente, una posición estimada del nodo tiene comunicación con nodos que no se encuentran en su radio de comunicación.

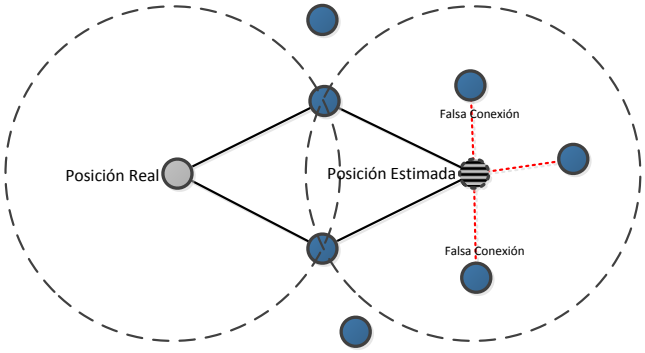


Figura 3.1 Falsa conexión entre nodos.

3.2 Generación de Escenario

Para las pruebas necesarias en los siguientes capítulos se debe contar con las posiciones de los nodos anclas y las distancias entre nodos vecinos. Por falta de alguna arquitectura real de pruebas se optó por generar las WSNs de forma aleatoria utilizando algún algoritmo.

Se diseñó un algoritmo en lenguaje C++ que permite distribuir de forma aleatoria una cierta cantidad de nodos sobre un área delimitada, y además obtiene la distancia entre ellos.

Una vez calculadas las distancias entre nodos, cada distancia obtenida se compara con el radio de comunicación definido, con el fin de obtener los vecinos de cada nodo y así lograr agruparlos.

Con el objetivo de que la simulación sea lo más real posible, a cada distancia entre nodos vecinos se les aplica un error de distancia. Es decir, las medidas de distancias entre nodos vecinos obtenidas con el algoritmo, son degradadas con ruido Gaussiano (ecuación 3.2), luego las nuevas distancias serán las distancias medidas "δ"(simularan una distancia obtenida con algún método de la sección 2.2.2).

$$\delta'_{a,b} = \delta_{a,b}(1.0 + RG \cdot FR) \tag{3.2}$$

Donde $\delta_{a,b}$ y $\delta'_{a,b}$ son la distancia real y la distancia medida respectivamente, entre dos nodos a y b. FR es el factor de ruido, con un valor del 10%, y el ruido Gaussiano (RG) tiene una media 0 y una desviación estándar de 1.

Finalmente los datos obtenidos con el algoritmo propuesto deben ser guardados en un vector y una matriz, ver 3.3. En el vector serán almacenados los datos correspondientes a las posiciones de los nodos anclas, y en la matriz las distancias entre nodos vecinos. Además, la matriz generada contará con las siguientes características:

1. Al ser una matriz simétrica sólo es necesario utilizar la matriz triangular superior de ésta (es posible por la condición de simetría en la comunicación entre nodos).
2. En la matriz triangular superior los nodos que no son vecinos, su distancia es definida como 0.

$$\beta = [x_1, y_1, x_2, y_2, \ldots, x_K, y_K] \qquad \text{y} \qquad \Delta := \begin{pmatrix} 0 & \delta_{1,2} & \delta_{1,3} & \delta_{1,4} & \cdots & \delta_{1,n} \\ 0 & 0 & \delta_{2,3} & \delta_{2,4} & \cdots & \delta_{2,n} \\ 0 & 0 & 0 & \delta_{3,4} & \cdots & \delta_{3,n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \tag{3.3}$$

CAPÍTULO 4

Algoritmo Propuesto y su Implementación

El presente capítulo tiene por objetivo indicar la metodología aplicada para resolver el problema de optimización, presentado en el capítulo 3, que corresponde a la localización de nodos sensores.

Como se justificó en la sección 2.2.3.3 el problema se encuentra catalogado cómo NP-Complete, es decir, es difícil encontrar una solución exacta. Por consiguiente es recomendable utilizar alguna metaheurística que permita encontrar alguna solución óptima. Como el objetivo de la tesis está enmarcado en la implementación de un algoritmo genético, se utilizara ésta herramienta para resolver el problema de optimización.

4.1 Algoritmo Genético Secuencial Propuesto

La solución del problema, se basa en encontrar las posiciones de los nodos y en recrear la topología de la red que satisfaga todas las restricciones del problema al mínimo error. El algoritmo genético propuestos evalúa los distintos individuos creados de forma seudo-aleatoria, para luego seleccionar a los posibles candidatos a la solución del problema. Se debe recalcar que los algoritmos genéticos permiten encontrar soluciones cercanas al óptimo del problema.

El pseudocódigo para este algoritmo se muestra en Algoritmo 4.1. La estructura general de un algoritmo genético hace uso de un operador de selección que toma individuos desde la población (línea 4), un operador de cruce y mutación que producen nuevos individuos (líneas 5 y 6), y un operador de reemplazo que elige los individuos para la siguiente generación (línea 8).

Algoritmo 4.1: Pseudocódigo de un algoritmo genético secuencial

Datos: Criterio de convergencia "*EndCoding*", probabilidad de cruce, mutación y selección;
Resultado: Mejor solución;

```
1  Inicialización de la población (P_t) y contador t = 0;
2  Evaluar(P_t) ;
3  Mientras no se cumpla EndCoding hacer
4      Padres ← operadorSelección(P_t);
5      Hijos ← operadorCruce(Padres);
6      Mutación(Hijos) ;
7      Evaluar(Hijos) ;
8      P_{t+1} ← reemplazo(Hijos,P_t);
9      P_t ← P_{t+1};
10     t ← t + 1;
11 Fin
```

Como se explicó en la sección 2.3, el algoritmo genético secuencial ("AGS") cuenta con varias etapas, de las cuales se destacan:

- Creación de la población inicial.
- Cálculo del fitness para cada individuo de la población
- Operadores genéticos de cruzamiento y mutación.
- Métodos de selección y reemplazo.

A continuación, se explica en detalle las etapas necesarias para la implementación del algoritmo genético secuencial y paralelo.

4.1.1 Representación de los Individuos

Los individuos se pueden representar de muchas maneras, pero se debe seleccionar de forma correcta, ya que de esta dependerán las siguientes etapas del algoritmo. Para éste problema la representación escogida utiliza un vector con números reales como el propuesto por Molina (2010).

Una solución candidata es un vector que almacena las coordenadas bi-dimensionales de los nodos desconocidos de la WSN. Primero que todo, los nodos de la red deben ser enumerados, luego todos los individuos generados en el algoritmo tendrán la siguiente característica, ser un vector con números reales y con largo igual al doble de nodos presentes en la red. Los dos primeros elementos corresponden a las coordenadas x y y del primer nodo, desde este punto cada dos posiciones del vector se representarán las coordenadas x y y de los siguientes nodos. La Figura 4.1 ilustra una solución codificada (individuo) para éste problema.

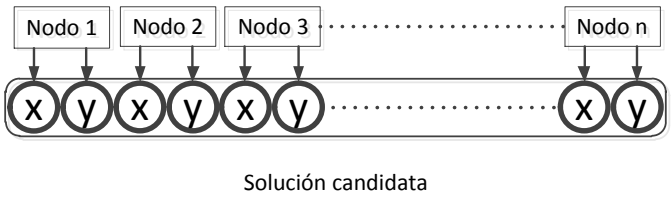


Figura 4.1 Solución codificada para el problema de localización, según (Molina, 2010)

Para simplicidad del problema las primeras posiciones del vector corresponden a las posiciones de los nodos anclas, y estas posiciones pueden tomar cualquier valor arbitrario ya que no se utilizan para las etapas posteriores.

4.1.2 Operadores Genéticos

Los operadores genéticos dependen principalmente de la codificación de los individuos, para los individuos de una población generalmente se les aplican los operadores de cruce y mutación. Cada uno de éstos operadores tiene un objetivo diferente, para el caso del operador

de cruce su objetivo es intensificar la búsqueda, por el contrario, el operador de mutación diversifica la búsqueda.

A continuación se describen los operadores genéticos utilizados para la resolución del problema de localización de nodos.

4.1.2.1 Operador de Cruce

Como se definió en la revisión bibliográfica el operador de cruce se utiliza para intercambiar genes entre dos o más cromosomas de una misma población. El operador de cruce que se utilizará, es el *simulated binary crossover* (para mayor información consultar la sección 2.3.3.6), ya que es sugerido por Molina & Alba (2011). Un pseudocódigo para éste operador se muestra en Algoritmo 4.2.

Algoritmo 4.2: Pseudocódigo del operador SBX

Datos: Dos individuos padres, total nodos anclas "K", total nodos "n";
Resultado: Dos individuos hijos;

Inicialización de los parámetros $r1$, $r2$, $\overline{\beta}$, u, n_c;
Para $i \leftarrow 2 \cdot K$ **hasta** $2 \cdot n$ **hacer**
  Generar número aleatorio u entre $[0,1]$;
  Si $u \leq \frac{1}{2}$ **entonces**
    $\overline{\beta} = 2u^{\frac{1}{n_c+1}}$;
  en otro caso
    $\overline{\beta} = \frac{1}{2(1-u)}^{\frac{1}{n_c+1}}$;
  Fin
  Guardar $individuo1[i]$ en $r1$;
  Guardar $individuo2[i]$ en $r2$;
  // Se generan los individuos hijos
  $individuo1[i] = \frac{1}{2}\left[\left(1+\overline{\beta}\right)r1 + \left(1-\overline{\beta}\right)r2\right]$;
  $individuo2[i] = \frac{1}{2}\left[\left(1-\overline{\beta}\right)r1 + \left(1+\overline{\beta}\right)r2\right]$;
Fin

4.1.2.2 Operador de Mutación

El operador de mutación juega un papel importante durante la evolución, ya que está encargado de introducir un grado de diversidad a cada generación. El operador estará formado por dos operadores de mutación utilizados de forma independiente en otros trabajos, éstos son: mutación por intercambio "Swap" (descrito en la sección 2.3.3.7) y single vertex neighborhood "SVN"(propuesto por Zhang et al. (2008)).

El operador de intercambio es muy utilizado en la literatura, y se aplica normalmente en los problemas de permutaciones. Como se explicó en la sección 2.3.3.7 es un operador que inicialmente fue propuesto para representaciones binarias. Consiste en elegir aleatoriamente dos genes de un individuo para luego intercambiarlos.

El operador SVN es un operador diseñado especialmente para el problema de localización. Él operador selecciona un vértice y luego lo mueve a algún punto de un circulo de radio decreciente (ver Figura 4.2). Por ejemplo, si se selecciona un vértice $v_i(x_i, y_i)$ para la mutación, entonces la nueva coordenada (x_i', y_i') de v_i estará definida por la Formula 4.1.

$$\begin{aligned} x_i' &= x_i + r\cos(\theta) \\ y_i' &= y_i + r\sin(\theta) \end{aligned} \tag{4.1}$$

Donde el radio $r = DisIdeal \cdot \frac{1-t}{T}$; $DisIdeal = \sqrt{\frac{s}{n}}$ es la distancia deseada entre nodos vecinos, $s = (b-a)^2$ es el área aproximada de la región en el plano, n es el número de nodos sensores, t es la generación actual, T es el número máximo de generaciones y por último $\theta \in [0, 2\pi]$ es un ángulo generado de forma aleatoria. Como se puede observar, el radio disminuye conforme el algoritmo progresa.

La ventaja que proporciona el operador de intercambio es su convergencia prematura, lo que permite llegar de forma más rápida a un valor de fitness, pero tiene como desventajas quedar atrapado en un óptimo local. En cambio el operador especifico es más lento, por el motivo de que no da grandes saltos en el espacio de búsqueda, pero tiene como ventaja el no quedar atrapados en óptimo locales. Con el objetivo de obtener las ventajas de ambos

operadores se utiliza un encapsulamiento, el cual permite definir la probabilidad de que se aplique cada uno de los operadores a una cierta población. Los pseudocódigos para cada operador se muestran en Algoritmo 4.3 y Algoritmo 4.4.

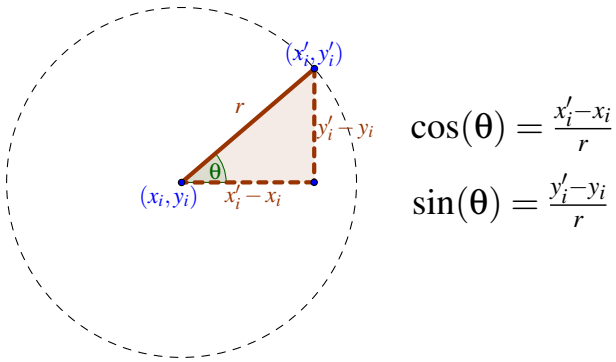


Figura 4.2 Operador single vertex neighborhood

Algoritmo 4.3: Pseudocódigo del operador de intercambio

Datos: Un individuo padre, y cantidad de intercambios "swap";
Resultado: Un individuo hijo mutado;

```
Inicialización de los parametros gen1, gen2;
Para i ← 0 hasta i ≤ swap hacer
    // Generar número aleatorio entero desde un intervalo [0,m], donde m es el tamaño del individuo
    gen1 = randomInteger(individuo.size()) ;
    Hacer
        gen2 = randomInteger(individuo.size());
    mientras gen1 == gen2;
    Intercambiar el valor de individuo[gen1] con el valor de individuo[gen2] ;
Fin
```

Algoritmo 4.4: Pseudocódigo del operador single vertex neighborhood

Datos: Un individuo padre, total nodos anclas "K", total nodos "n", total de generaciones "T", DisIdeal, generación actual "t";
Resultado: Un individuo hijo mutado;

```
Inicialización de los parametros θ, radio;
radio = DisIdeal · (1 − t/T) ;
Para i ← K hasta i < n hacer
    θ = random.uniform(0, 2π);
    individuo[i × 2] = individuo[i × 2] + r cos(θ);
    individuo[i × 2 + 1] = individuo[i × 2 + 1] + r sin(θ);
Fin
```

4.1.3 Función de Evaluación

En la implementación de la función de fitness o evaluación, se utilizará la función formulada (ecuación 3.1) en la sección 3.1.1. Para la implementación de ésta se deben utilizar los datos 3.3, que corresponden a las posiciones de los nodos anclas y las distancias entre nodos vecinos. Utilizar solo ésta función como función de evaluación podría provocar una gran cantidad de individuos infactibles. Con el fin de disminuir éstos individuos se hace uso de las restricciones descritas en la sección 3.1.2.

Como se describió en la sección 2.3.3.4 existen variadas técnicas para resolver el problema de soluciones infactibles. Para la resolución del problema planteado la técnica seleccionada es por medio de una función de penalización. Dicha función consiste en modificar la función objetivo, sumándole o restándole un término que depende de las restricciones violadas. De esta forma, el problema se transformará a uno sin restricciones.

La restricción de delimitación del área de trabajo es considerada como una restricción fuerte, ya que sin ésta restricción se podrían generar coordenadas fuera de rango, y en consecuencia individuos infactibles que podrían ser considerados en la población. Para la implementación de ésta restricción se utiliza un generador de números aleatorios entre los rangos permitidos (a,b). Con él se inicializan los genes de cada individuo, además de restringir los genes generados en los operadores genéticos.

Desde la matriz de distancia se puede inferir qué nodos están o no comunicados. Para la restricción de falsos vecinos se procede a calcular la distancia entre los nodos que no deberían estar comunicados, si ésta distancia es menor que el radio de comunicación se contabiliza como restricción no cumplida. El individuo con menor cantidad de conexiones falsas tendrá un menor fitness, y por lo tanto una mayor probabilidad de ser seleccionado.

La función de fitness 4.2 es una combinación lineal positiva de la función objetivo propuesta en la sección 2.3.3.4 definida como F, y de la restricción de falsos vecinos definida como f_1. w_0 y w_1 son los pesos correspondientes. La idea detrás del peso es obligar al algoritmo, atacar todos los términos por igual o dar mayor prioridad a alguno. Por ejemplo, si F toma valores entre $[0,100]$ y f_1 valores en $[0,1]$, entonces se propone $\Phi = 0.05F + 50f_1$, de esta manera los dos términos estarán entre 0 y 50, y el AG estará obligado a atacar a los dos términos por igual.

$$\Phi = w_0 F + w_1 f_1 \tag{4.2}$$

El objetivo de la función de fitness modificada Φ, es verificar que las soluciones consideradas cumplan con las restricciones del problema. En caso de no cumplirlas, el valor de fitness de las soluciones consideradas aumenta (ya que se busca disminuir el error). Por ejemplo, si sólo se considerara la función 3.1 sin las restricciones, una solución infactible podría tener el mismo valor de fitness que una solución factible, pudiendo ser descartada ésta última.

Por ultimo en Algoritmo 4.5 se muestra un pseudocódigo de la función de penalización, se puede apreciar que en la función de penalización existen tres parámetros a diferencia de los propuestos en 4.2, esto se realizó con el objetivo de darle un mayor peso a las coordenadas de los nodos cercanos a los nodos anclas, y con esto lograr obtener posiciones de mejor calidad.

Algoritmo 4.5: Pseudocódigo de la función de penalización

Datos: Un individuo "X", total nodos anclas "K", total nodos "n", matriz de distancia "Δ", posiciones nodos anclas "β", radio de comunicación "r";
Resultado: Fitness del individuo;

```
// Contador de restricciones no cumplidas "count" y distancia calculada "DC"
Inicialización de los parámetros Error1, Error2, Φ,"count" y "DC";
Para i ← 0 hasta i < K hacer // Utiliza el vector de posición
    Para j ← K hasta j < n hacer
        Si δ[i, j] ≠ 0 entonces
            DC = √((β[i×2] − X[j×2])² + (β[i×2+1] − X[j×2+1])²) ;
            Error1 =| DC − δ[i, j] | +Error1 ;
        en otro caso
            DC = √((β[i×2] − X[j×2])² + (β[i×2+1] − X[j×2+1])²);
            Si DC ≤ r entonces
                count = count + 1;
            Fin
        Fin
    Fin
Fin
Para i ← K hasta i < n hacer // Sólo utiliza el vector individuo
    Para j ← K hasta j < n hacer
        Si δ[i, j] ≠ 0 entonces
            DC = √((X[i×2] − X[j×2])² + (X[i×2+1] − X[j×2+1])²) ;
            Error2 =| DC − δ[i, j] | +Error2 ;
        en otro caso
            DC = √((X[i×2] − X[j×2])² + (X[i×2+1] − X[j×2+1])²);
            Si DC ≤ r entonces
                count = count + 1;
            Fin
        Fin
    Fin
Fin
// Función de penalización, los pesos se obtienen a través de pruebas
Φ = w0 × Error1 + Error2 + w1 × count ;
```

4.1.4 Motor Evolutivo

El término motor evolutivo denota las diferentes partes de un algoritmo evolutivo que simulan la evolución darwinista. Esta idea toma lugar en dos etapas diferentes de un algoritmo genético:

- La selección como se definió en la sección 2.3.3.5, es la etapa que tiene por objetivo decidir cuáles son los individuos más aptos de la población.

- El reemplazo ocurre después de la creación de todos los hijos por medio de los operadores genéticos. Los nuevos individuos se convierten en los padres de la siguiente

generación.

ParadisEO cuenta con tres métodos de selección, los cuales son: selección por ruleta, torneo estocástico binario y selección por torneo determinista. Los métodos por torneo son fáciles y rápidos de implementar, seleccionan a un individuo dentro de una población basándose en una simple comparación (usualmente por medio del fitness). Para este trabajo, el método seleccionado es el torneo determinista.

Dentro de ParadisEO el tamaño de la población se supone constante desde una generación a otra, aunque es posible saltarse esta restricción. Para el operador de reemplazo se decide mantener una población constante. El operador seleccionado se encarga de mezclar a los padres con sus hijos, además de hacer uso del elitismo (ver sección 2.3.3.8) que permite mantener a los mejores padres en la descendencia.

4.1.5 Criterio de Convergencia

El criterio de parada seleccionado desde la sección 2.3.3.9, es el máximo número de generaciones, debido principalmente a que el operador de mutación single vertex neighborhood necesita conocer previamente el número total de generaciones que se ejecutarán.

4.2 Algoritmo Genético Paralelo

Para la implementación de un algoritmo genético paralelo ("AGP"), ParadisEO cuenta con los módulos PEO y SMP, los cuales permiten que los usuarios se despreocupen de cuestiones relacionadas con el paralelismo, puesto que cuenta con templates preparados para utilizar las correspondientes librerías de paralelismo.

La arquitectura de los algoritmos propuestos tienen como base al sistema operativo Linux "Debian". Para la comunicación entre los procesadores se hace uso de la librería "thread" que se encuentra disponible en las nuevas características del estándar C++ 2011.

Nótese que en la Figura 4.3 se ha presentado a una parte de los AGP sobre ParadisEO y a otra parte por fuera, para indicar que algunos de los componentes de un AGP son provistos por ParadisEO y otros deben ser provistos por el programador.

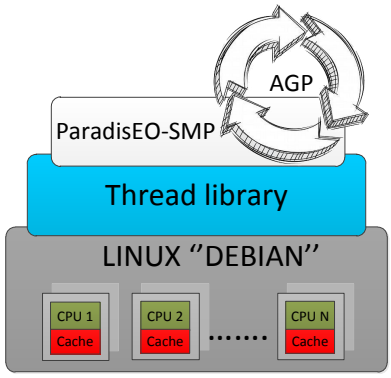


Figura 4.3 Arquitectura de la solución propuesta.

Los algoritmos implementados y descritos a continuación hacen uso de la misma representación, operadores genéticos, función de evaluación y criterio de convergencia descritos en la Sección 4.1.

4.2.0.1 Implementación Maestro/Esclavo Bajo Arquitectura SMP

El modelo maestro/esclavo ("AGME") desarrollado, se diferencia del algoritmo secuencial principalmente en la forma de calcular la función de fitness y en la ejecución de los operadores genéticos, ya que se evalúan de forma paralela varios individuos haciendo uso de los correspondientes núcleos o hilos esclavos. El funcionamiento de esta estructura depende de un núcleo maestro el cual se encarga de enviar los datos a procesar hacia los hilos esclavos. El papel del maestro es a menudo asociado con el programador que distribuirá y gestionará las comunicaciones de los núcleos esclavos.

Con este método de paralelización no hay garantía de obtener mejores individuos en comparación con el algoritmo secuencial. Sin embargo, lo que ocurre es que se ejecutan más

iteraciones del bucle genético en el mismo espacio de tiempo, y por tanto, se llega a buenas soluciones en un menor tiempo.

4.2.0.2 Implementación Modelo de Islas Bajo Arquitectura SMP

El modelo de islas ("AGisla") como se describió en la sección 2.4.4 tiene por objetivo ejecutar de forma simultanea varios algoritmos genéticos, y que además cuenten con la capacidad de comunicarse entre sí. Para la implementación del algoritmo se debe especificar una topología de comunicación entre las islas y políticas de intercambio de individuos.

Cada núcleo está encargado de evolucionar una respectiva población, haciendo uso para ello de los correspondientes operadores. El número de hilos del algoritmo dependerá de la cantidad de núcleos disponibles en la arquitectura. Para el intercambio de los individuos se deben definir políticas de migración e inmigración. Estas políticas consisten en definir cada cuantas generaciones se deben intercambiar individuos, criterios de selección y criterios de reemplazo.

La comunicación entre las islas es una de las partes más importantes de este algoritmo, ya que si no existiera no podría haber intercambio de individuos entre ellas. Principalmente depende de la topología que se utilice, actualmente se pueden utilizar arquitecturas tipo estrella, anillo, grafo completo e hipercubo, aunque existe la posibilidad de implementar la topología que uno estime conveniente.

Este tipo de implementación permite contar con una gran diversidad de individuos, ya que por cada isla habrá una población que evolucionará de forma independiente, lo cual permite explorar un mayor espacio de búsqueda y además evitar una convergencia prematura.

El pseudocódigo para este algoritmo se muestra en Algoritmo 4.6. La estructura general de un algoritmo genético hace uso de un operador de selección que toma individuos desde la población (línea 5), un operador de cruce y mutación que producen nuevos individuos (líneas 6 y 7), y un operador de reemplazo que elige los individuos para la siguiente generación (línea 9). Entre las líneas 10 y 16 se describen las políticas de migración e inmigración, donde la

línea 12 define la topología a utilizar, además se define el operador de selección de los migrantes (línea 13) y el operador de reemplazo de los inmigrantes (línea 14).

Algoritmo 4.6: Pseudocódigo de un algoritmo genético paralelo

Datos: Criterio de convergencia "*EndCoding*", probabilidad de cruce, mutación, selección, selección de los migrantes, generación de intercambio de individuos "*EachGen*";
Resultado: Mejor solución para cada isla;

```
1  ISLA^I // I = 1, 2, ..., N donde N es el número de núcleos;
2  Inicialización de la población P_t^I y contador t = 0;
3  Evaluar(P_t^I) ;
4  Mientras no se cumpla EndCoding hacer
5  |  Padres ← operadorSelección(P_t^I);
6  |  Hijos ← operadorCruce(Padres);
7  |  Mutación(Hijos) ;
8  |  Evaluar(Hijos) ;
9  |  P_{t+1}^I ← reemplazo(Hijos,P_t^I);
10 |  P_t^I ← P_{t+1}^I;
11 |  Mientras se cumpla EachGen hacer // Politicas de migración e inmigración;
12 |  |  P_t^I ∪ ISLA^J :: P_t^J // Interacción con los vecinos
13 |  |  migrantes ← operadorSelección (ISLA^J :: P_t^J);
14 |  |  P_{t+1}^I ← reemplazo(migrantes,P_t^I);
15 |  |  P_t^I ← P_{t+1}^I;
16 |  Fin
17 |  t ← t + 1;
18 Fin
```

4.3 Implementación computacional

Para la implementación computacional de los algoritmos descritos en las secciones anteriores es necesario utilizar el framework ParadisEO (para su instalación consultar anexo A), pues cuenta con librerías que facilitan el desarrollo de los distintos algoritmos. Además, para la creación de los ejecutables es necesario utilizar *CMake*, el cual se encarga de controlar el proceso de compilación, en conjunto con éste es necesario contar con un compilador de C++.

El compilador dependerá del sistema operativo utilizado. Para el caso de Windows se puede hacer uso de *MinGW* (es una adaptación del compilador gcc de Linux) o *visual C++*. En caso de Linux se hace uso de los compiladores *gcc* y *g++*.

El algoritmo secuencial puede ser implementado sobre plataformas Windows y Linux, mientras que los algoritmos paralelos solo pueden ser implementados sobre arquitectura Linux, debido a las restricciones de la librería SMP de ParadisEO.

La interacción del usuario con el programa es por medio de archivos con extensión ".param" y texto plano, los cuales permiten el fácil ingreso de los distintos datos y parámetros. Por otro lado, los datos de salida se obtienen por medio de archivos de texto plano.

La implementación se puede dividir en cuatro grupos principales, descritos como:

- Generación o ingreso de escenario.
- Datos y parámetros.
- Metaheurística (descrita en las secciones anteriores).
- Datos de salida.

El diagrama de flujo de la Figura 4.4 muestra cada etapa de la implementación computacional.

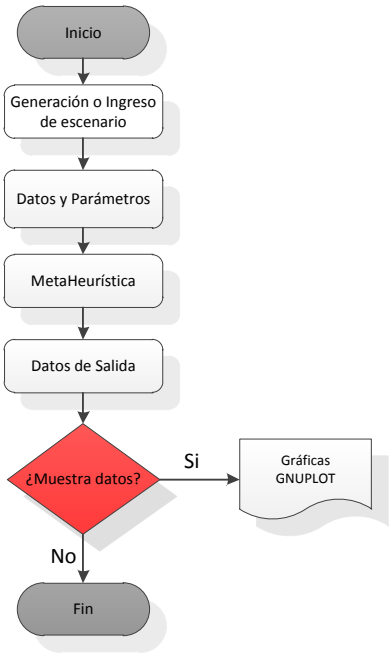


Figura 4.4 Diagrama de flujo de las principales etapas de la implementación.

4.3.1 Generación o Ingreso de Escenario

Para la evaluación de los algoritmos propuestos es necesario contar con los datos de posición de los nodos anclas y distancias estimadas entre nodos comunicados de una red inalámbrica de sensores (WSN). Estos datos pueden ser proporcionados por medio de archivos de textos planos o generados de forma aleatoria, como se describió en la Sección 3.2.

Sí se cuenta con un escenario real los datos deben ser proporcionados por medio de archivos independientes con nombres *"Anclas.txt"* y *"Matriz.txt"*. El primero de ellos está formado con la posición de los nodos anclas. Su estructura está constituida por dos columnas (posiciones x e y) y k filas, donde k es la cantidad de nodos anclas. El segundo archivo está formado por la distancia entre nodos, su estructura es una matriz cuadrada de tamaño n, donde n es el total de nodos. La cantidad mínima de datos que se deben proporcionar para ésta opción son; la delimitación del área de trabajo, la cantidad de nodos anclas y el total de

nodos (anclas y desconocidos).

Contar con un escenario real que esté formado por cientos de nodos es muy costoso. Para la solución de éste inconveniente existe la opción de generar el escenarios de forma aleatoria y para esto el algoritmo debe contar con los siguientes datos; delimitación del área de trabajo, cantidad de nodos anclas, total de nodos, radio de comunicación y semilla para la generación de los números aleatorios.

4.3.2 Datos y Parámetros

El ingreso de los parámetros se puede realizar por medio de un archivo ".param", donde se ingresan los datos correspondientes al escenario y los datos de las distintas rutinas que se implementan.

En el archivo de configuración se incluyen valores por defecto para los distintos parámetros, los cuales pueden ser modificados por el usuario. para ajustarse a los requerimientos de cada proyecto.

A continuación se describen algunos de los principales datos y parámetros de entrada, los cuales son agrupados según la rutina que los emplea. Algunos de los datos de entrada son:

- El parámetro "Escenario" especifica si el escenario será generado de forma aleatoria o se ingresarán los datos.
- "PopSize" es el número de individuos de la población para los algoritmos.
- "MaxGen" es el criterio de parada, el cual define el número máximo de generaciones, es un parámetro importante para el operador de mutación *single-vertex-neighborhood* "SVN" propuesto por (Zhang et al., 2008).
- "Pcruza", probabilidad de cruzamiento ($0 \leq P_c \leq 1$) para el operador "SBX", en el orden de $0,87$.
- "Pmutation", probabilidad de mutación ($0 \leq P_m \leq 1$) para el encapsulamiento de los operadores SVN y Swap (intercambia dos componentes de un cromosoma).

- "SizeTorneo" especifica el tamaño del torneo para seleccionar los individuos que se utilizarán en la reproducción.
- "sizeElist" define la cantidad de individuos que pasarán a la siguiente generación sin sufrir modificaciones, son seleccionados los mejores individuos.
- "sizeTorneo1" especifica el tamaño del torneo para seleccionar los individuos que pasarán a la siguiente generación, se ejecuta con el operador de reemplazo elitista.

El modelo de islas cuanta con algunos parámetros especiales que son los encargados de las políticas de migración e inmigración de los individuos.

- El parámetro "Intercambio" define cada cuantas generaciones migrarán los individuos.
- "torneoIsla" tamaño del torneo para seleccionar los individuos a migrar.
- "selecIsla" es la cantidad de individuos que migrarán.

Los parámetros asociados a los algoritmos son ajustados de acuerdo al tipo de problema, y se explora el mejor valor para cada parámetro según sea el resultado del algoritmo.

En el Anexo C se puede apreciar el archivo de configuración de los algoritmos, en el cual se deben ingresar los datos del escenario y su configuración.

4.3.3 Datos de Salida y Despliegue de resultados

La visualización de los resultados encontrados por los algoritmos se presenta en dos formatos. El primero de ellos corresponde a archivos de texto plano llamados "generacion.sav" y "stats.xg", en el primero se muestran todos los individuos de la correspondiente generación, éste archivo se genera cada una cierta cantidad de generación que es definida con anterioridad en el archivo de configuración. En el segundo archivo se muestra por cada generación el tiempo total de ejecución, mejor individuo, media y desviación estándar. El segundo formato corresponde a las visualizaciones gráficas, entre las que se puede nombrar: la evolución del mejor candidato, el plano x e y del mejor individuo y, por último, la diversidad del algoritmo. Para la representación gráfica se hace uso del software "Gnuplot".

CAPÍTULO 5

Pruebas y resultados

En el siguiente capítulo se describe la metodología adoptada para los experimentos, y materiales utilizados durante el desarrollo del presente trabajo de título. Para ello, se comienza detallando las especificaciones técnicas del hardware utilizado en la ejecución de los algoritmos diseñados, indicando las versiones de todas las aplicaciones y compiladores utilizados. A continuación, se describe el diseño de las pruebas que permiten evaluar el comportamiento de los algoritmos propuestos. Para ello se define el set datos, configuración paramétrica, las métricas de desempeño, y los tipos de gráficos utilizados para la representación de los datos. Por último, se presentan y discuten los resultados obtenidos de forma individual y haciendo comparativas entre los algoritmos. Además, se describe la metodología estadística que permite comprobar que las diferencias existentes entre los resultados obtenidos son estadísticamente relevantes.

5.1 Especificaciones de Hardware y Software

El hardware utilizado para el desarrollo, implementación y prueba de los algoritmos desarrollados cuenta con las siguientes características:

- $Intel^{®}$ $Core^{TM}$ i3 CPU 540 @3,06GHz
- 2GB de RAM
- HD 500GB

Este procesador cuenta con la tecnología Hyper-Threading lo que significa que por cada núcleo físico puede ejecutar dos cadenas de procesamiento, éstos son conocidos como hilos de ejecución. Finalmente el procesador está formado por dos núcleos físicos y cuatro núcleos virtuales.

Todos los experimentos se realizaron sobre el sistema operativo Debian wheezy en el cual se instalaron y configuraron los siguientes software:

- Compilador gcc 4.7.2 y g++ 4.7.2
- CMake version 2.8.9
- Boost C++ 1.49.0.1
- ParadisEO 2.0.1
- Gnuplot 4.6 patchlevel0
- Entorno de programación Code::Blocks 10.05
- R Project

5.2 Diseño de Experimentos

El primer paso para el diseño de experimentos es establecer un conjunto de escenarios que permitan evaluar el comportamiento de los distintos algoritmos propuestos. Además, es necesario definir las características del entorno de pruebas, como la cantidad de generaciones y el número de ejecuciones.

Debido a la naturaleza estocástica de los algoritmos estudiados, las comparaciones entre ellos en función de una sola ejecución no es consistente, por lo que se deben realizar sobre un gran conjunto de resultados obtenidos tras realizar un elevado número de ejecuciones independientes de los algoritmos sobre los problemas planteados.

Una vez obtenidos los resultados de las respectivas ejecuciones se debe realizar un análisis estadístico que permita afirmar que los datos obtenidos son estadísticamente significativos y que no son fruto de la aleatoriedad, luego de esta evaluación los datos serán comparados empíricamente.

5.2.1 Escenarios de Pruebas

De un total de diez escenarios generados a partir del algoritmo propuesto en la sección 3.2, se seleccionó solo uno en función de la distribución de los nodos inalámbricos (lo más disperso posible), desde este escenario se modificó el radio de comunicación entre nodos, obteniendo cuatro escenarios con distintos enlaces de comunicación. Las principales propiedades (número de nodos, número de enlaces y radio de comunicación) de la instancia seleccionada se muestran en la Figura 5.1 y Tabla 5.1.

El objetivo de estas pruebas es evaluar el comportamiento de los algoritmos, ya sea en el rendimiento de estos y en su convergencia, además de mostrar la influencia en la cantidad de enlaces de comunicación.

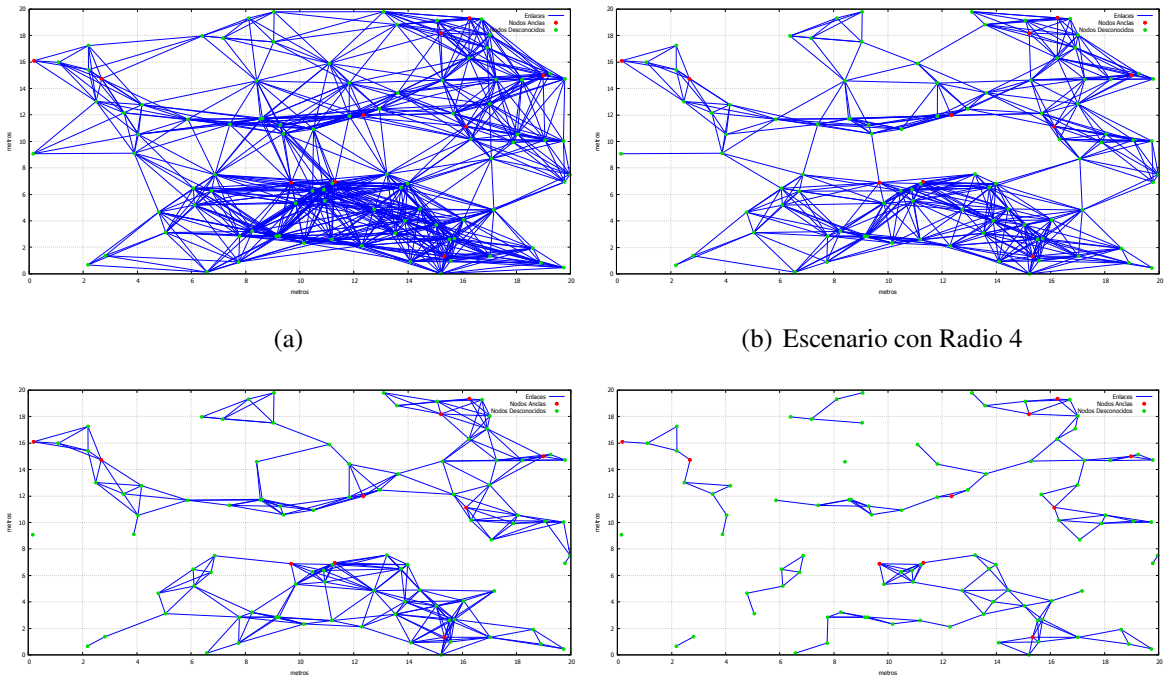
Se estableció un número pequeño de nodos anclas, que funcionarán como puntos de referencias para el resto de nodos. La cantidad por defecto de nodos anclas seleccionada para todas las instancias, se encuentra alrededor del 10% del total de nodos presentes en la red.

Tabla 5.1: Características de los escenarios.

Escenarios	R5 5.1(a)	R4 5.1(b)	R3 5.1(c)	R2 5.1(d)
Número de nodos	100	100	100	100
Número de enlaces	808	521	308	160
Radio max. Comunicación	5	4	3	2

Se debe recalcar que existe completo conocimiento de las posiciones reales de todos los nodos presentes en la red, sin embargo, los algoritmos de optimización no tienen acceso a esta

información. Solo es de utilidad para la posterior verificación de las soluciones obtenidas por los algoritmos.



(a) (b) Escenario con Radio 4

(c) Escenario con Radio 3 (d) Escenario con Radio 2

Figura 5.1 Escenario con distintos radios de comunicación.

5.2.2 Parámetros de Configuración

La configuración paramétrica de los algoritmos fue ajustada de forma empírica. Los valores de los parámetros fueron obtenidos luego de 20 ejecuciones independientes de 10,000 generaciones cada una, en la Tabla 5.2 se pueden apreciar los distintos parámetros y su configuración.

Tabla 5.2: Configuración Paramétrica (a)Algoritmo Genético Secuencial y Maestro/Esclavo; (b)Modelo de Islas.

Algoritmo	AG Secuencial	Maestro/Esclavo
Evaluaciones	60.000	
Población	100	
Codificación	Real	
Selección	Torneo Determinista	
Reemplaza	Elitismo	
Cruza	SBX, $P_c = 0.87$	
Mutación	SVN y Swap, $P_m = 0.85$	
Trabajadores	-	4

(a)

Algoritmo	AG Modelo Islas
Islas	2 y 4
Evaluaciones	60.000
Población	100
Selección	Torneo Determinista
Reemplaza	Elitismo
Cruza	SBX, $P_c = 0.87$
Mutación	SVN y Swap, $P_m = 0.85$
Intercambio	7500 y 5000
Sel. Migración	Torneo Determinista

(b)

5.2.3 Configuración del Entorno de Pruebas

Para la comparación de los resultados se ha considerado como condición de parada un máximo de 60,000 generaciones, y se hace uso de las configuraciones indicadas en la Tabla 5.2. Además, para obtener los resultados de los experimentos se realizaron a lo menos 26 ejecuciones por cada par algoritmo y escenario (hay una instancia con 4 configuraciones de enlaces, lo que implica tener un mínimo de 520 ejecuciones independientes), aunque en la mayoría de los casos, el número de ejecuciones independientes realizadas superan aproximadamente las 50 por problema.

5.2.4 Análisis Estadístico

El análisis estadístico de los datos permite verificar que los resultados obtenidos son relevantes estadísticamente y no fruto de la causalidad. El primer paso para este estudio es realizar una análisis descriptivo de los datos, prestando especial atención a la presencia de puntos atípicos. Para una representación intuitiva de los resultados se utilizan las gráficas tipo diagrama de cajas "*Boxplot*".

La metodología seleccionada para el análisis global de los datos (Figura 5.2), es mediante un análisis de la varianza (test de ANOVA), o en caso de no cumplir con los requisitos del test de ANOVA, mediante un test no paramétrico (Kruskal-Wallis o Wilcoxon).

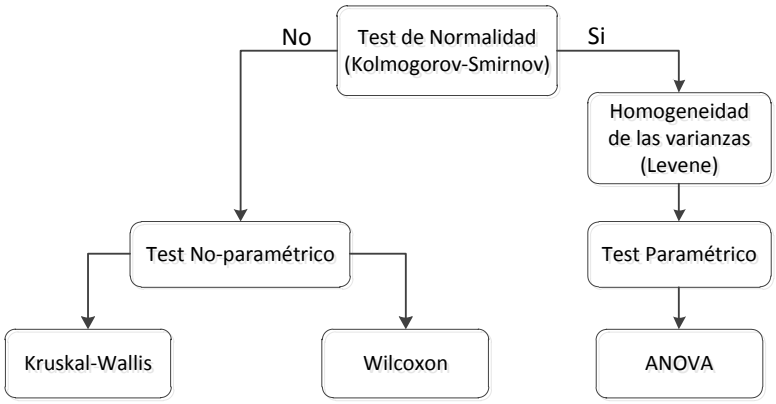


Figura 5.2 Análisis estadísticos de los resultados experimentales.

Para la implementación del test de ANOVA es necesario cumplir con los siguientes puntos (García et al., 2007):

1. Independencia de las variables.
2. Los datos obtenidos deben proceder de una población normal, para este trabajo se utilizará el test de Kolmogorov-Smirnov.
3. La varianza para las distintas poblaciones debe ser la misma. Esta propiedad es conocida como homocedasticidad. La verificación de esta propiedad se puede llevar acabo por medio del test de Levene.

En caso de no cumplir con alguno de los requisitos del test de ANOVA, se puede aplicar el test no paramétrico Kruskal-Wallis para la comparación de medianas. Este test permite verificar si los datos provienen de la misma población o son diferentes.

Una vez obtenidos los resultados del correspondiente test implementado, ya sea paramétrico o no paramétrico, de cumplirse la hipótesis nula se procede a realizar un test de comparaciones múltiples que tiene por objetivo identificar cuál de los algoritmos presenta la diferencia estadística.

5.2.5 Análisis Empírico

Luego de comprobar que los resultados de las distintas implementaciones son estadísticamente relevantes, se procede a realizar un análisis empírico. Para ello se realiza una comparación en función de las métricas de desempeño, las cuales son el *Speedup* (S_p) y la *Eficiencia* (E_p), para mayor información ver la sección 2.6.3. Se debe recalcar que se hace uso del *Speedup* débil, es decir, se comparará el tiempo de ejecución promedio del modelo paralelo con respecto al promedio del modelo secuencial.

Se evalúa el comportamiento del mejor fitness y el fitness promedio de cada algoritmo propuesto, con el fin de encontrar evidencia empírica de la convergencia de las distintas propuestas. Además, se muestran los mejores resultados obtenidos de forma gráfica para los distintos escenarios.

5.2.6 Tipos de gráficos

Los distintos tipos de gráficos son obtenidos por medio de Gnuplot, y son utilizados para la representación de los resultados obtenidos luego de las ejecuciones de los algoritmos. A continuación se describirán los tipos de gráficos utilizados.

- Se hace uso de gráficas tipo boxplot que presentarán el fitness de las 26 ejecuciones de cada escenario, para así realizar un análisis comparativo de los algoritmos.
- Gráficas del speedup y eficiencia media de los algoritmos paralelos.
- Gráfica de barra que muestre el tiempo de ejecución para cada algoritmo.
- Gráfica del fitness vs tiempo de ejecución.

- Gráfica del fitness vs generación, donde se comparará el comportamiento del mejor fitness de cada algoritmo por cada escenario.
- Gráfica de la diversidad vs generación, muestra la diversidad de la población para cada algoritmo.
- Se presenta una gráfica del error de localización, en donde se compara la localización original de los nodos con la posición encontrada por los respectivos algoritmos.

5.3 Experimentos

En esta sección se utilizarán herramientas estadísticas para realizar un análisis de los resultados obtenidos entre los distintos algoritmos propuestos, y con esto obtener evidencia estadística. Además, como se dispone de una gran cantidad de datos para cada algoritmo, un primer paso consistirá en presentar los datos de forma gráfica, para que éstos puedan ser vistos de forma resumida y ordenada.

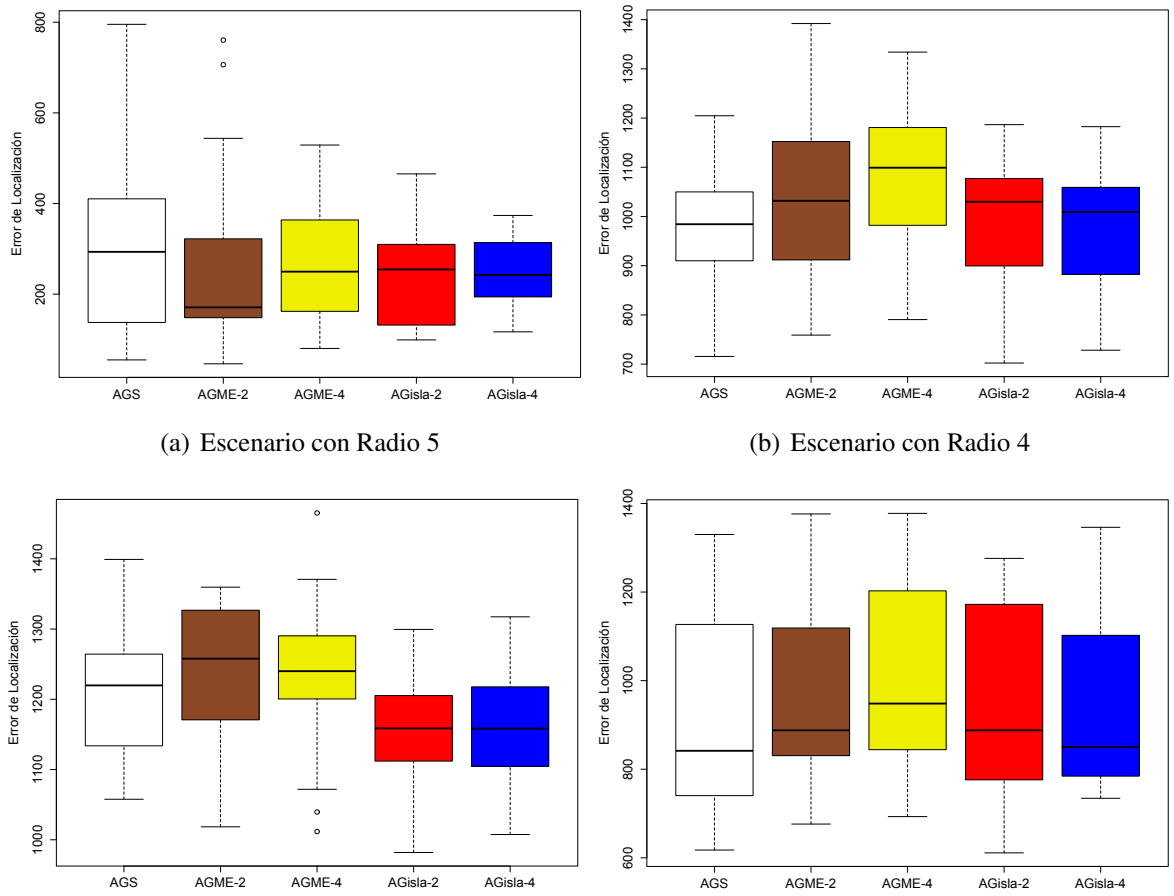
Luego se procede a realizar los análisis empíricos de los datos, como las métricas de desempeño, evaluación del tiempo de procesamiento, y por último se presentan de forma gráfica los valores de fitness obtenidos para los distintos algoritmos.

5.3.1 Visualización Gráfica de los Datos

Para la visualización de los datos se utilizó un diagrama de cajas (Boxplot), con el cual se estudia la distribución de los datos y la identificación de puntos atípicos. En la Figura 5.3 se representan los resultados obtenidos para las 26 ejecuciones independientes realizadas para cada par algoritmo-escenario.

Como se puede observar en la Figura 5.3, existen casos atípicos para los algoritmos AGME-2 y AGME-4 en los escenarios 5.3(a) y 5.3(c) respectivamente. Se decidió eliminar estos puntos del estudio estadístico, por ser considerados no representativos, por lo que el número de muestras ha disminuido para los casos descritos.

Los datos obtenidos por los algoritmos para el escenarios R5 se presentan en la Figura 5.3(a), se puede observar que los mejores valores de fitness (o error de localización) son obtenidos por los algoritmos AGS y AGME-2, aunque para el caso del AGS el 75% de los datos presentan una alta diversidad la que se encuentran entre el rango de 795,36 y 137,61. Por otro lado, el algoritmo que presenta una menor diversidad pero al mismo tiempo el peor fitness mínimo es el AGisla-4, de lo cual se puede concluir que existe una mayor probabilidad de encontrar valores de fitness entre los rangos 373,7 y 116,9.



(a) Escenario con Radio 5

(b) Escenario con Radio 4

(c) Escenario con Radio 3

(d) Escenario con Radio 2

Figura 5.3 Diagrama de cajas para los resultados de los distintos escenarios de pruebas.

Para el escenario R4 (Figura 5.3(b)), todos los algoritmos presentan aproximadamente

la misma diversidad, solo destacándose por su rango intercuartílico el algoritmo AGS. Los mejores errores de localización son obtenidos por los algoritmos AGS y AGisla-2.

En el escenario R3 (Figura 5.3(c)), el mejor fitness es 981,93 y es obtenido por el algoritmo AGisla-2. Al igual que el escenario anterior, la diversidad total para cada algoritmo es aproximadamente similar, solo destacándose por un menor rango intercuartílico los algoritmos AGisla-2 y AGisla-4.

Finalmente el escenario R2 (Figura 5.3(d)), obtiene los resultados con la mayor diversidad en el valor de fitness. Sólo destacan los algoritmos AGS y AGisla-2, los cuales obtuvieron los valores de fitness más bajos, 617,6 y 611,2 respectivamente.

Uno de los grandes problemas que se aprecia en los algoritmos propuestos son su alta diversidad, así como de sus límites superiores. En este contexto se puede observar que los algoritmos AGisla-2 y AGisla-4 presentan un comportamiento más estable y preciso en comparación con los demás algoritmos, esto se concluye debido a que presentan una menor desviación estándar.

5.3.2 Metodología Estadística

Para el análisis estadístico el primer paso es verificar si las muestras obtenidas son independientes, en este caso es clara la independencia, puesto que cada ejecución se realizó de forma independiente y con semillas generadas de forma seudo-aleatorias. El siguiente paso consiste en estudiar la normalidad de las observaciones obtenidas por cada algoritmo, para esto se hace uso del test Kolmogorov-Smirnov con la corrección de Lilliefors (Tabla 5.3), para todas las pruebas estadísticas se hace uso de un nivel de significancia de un 5%.

Tabla 5.3: Test de normalidad de Kolmogorov-Smirnov.

Algoritmos	P-value R5	P-value R4	P-value R3	P-value R2
AGS	0,4426	0,6099	0,834	0,0261*
AGME-2	0,004156*	0,9107	0,4702	0,03883*
AGME-4	0,72	0,4112	0,1265	0,00525*
AGisla-2	0,105	0,1138	0,5598	0,1556
AGisla-4	0,4044	0,1874	0,8833	0,01649*

"*" indica que no se cumple con la normalidad

Como se puede observar en la Tabla 5.3, en el escenario R5 el valor obtenido por el estadístico para el algoritmo AGME-2 es menor al valor de significancia, lo cual implica rechazar la hipótesis de normalidad. Para el caso del escenario R2 los algoritmos AGS, AGME-2, AGME-4 y AGisla-4 rechazan la hipótesis de normalidad. En el resto de escenarios no es posible rechazar la hipótesis nula de los datos.

Otro de los requisitos que se debe cumplir para aplicar un test paramétrico, es el de la homogeneidad de las varianzas. Esto se comprueba por medio del test de Levene (Tabla 5.4) basado en medianas.

Tabla 5.4: Test de Levene para la homogeneidad de varianzas.

	R5	R4	R3	R2
Estadístico de Levene	0,02559*	0,2422	0,5875	0,5875

"*" indica que las varianzas para un determinado escenario no son homogéneas

De acuerdo a los valores obtenidos por el estadístico de Levene, en los distintos escenarios, se puede concluir que la hipótesis de igualdad de varianzas para todas las estrategias desarrollas en los escenarios R4, R3 y R2 no se rechaza. En el caso del escenario R4 se rechaza la hipótesis nula, esto quiere decir que no es posible aplicar un test paramétrico, ya que no se cumple con el requisito de homogeneidad.

Finalmente, para la comparación de los resultados en los escenarios R4 y R3, se hará uso del test paramétrico ANOVA de una vía, ya que se cumple con los criterios de normalidad y homogeneidad de varianzas. En cambio, para los escenarios R5 y R2 se hará uso del test no paramétrico Kruskal-Wallis.

Para la implementación del test de ANOVA sobre los escenarios R4 y R3, primero se deben definir las hipótesis en estudio. En la Tabla 5.5 se define la hipótesis nula y alternativa.

Tabla 5.5: Planteamiento de Hipótesis para el test de ANOVA.

H_0	Los cinco algoritmos son idénticos en términos de la media del error de localización y las diferencias observadas son causa del azar.
H_1	Por lo menos un par de algoritmos no es idéntico en términos de la media del error de localización y las diferencias no son causa del azar.

Los resultados obtenidos tras la ejecución del test para los respectivos escenarios se muestran en las Tablas 5.6 y 5.7. Como se puede observar en ambos casos, el valor de significancia es menor a 0,05, existiendo diferencias significativas, por lo que se rechaza la hipótesis nula. Esto quiere decir que a lo menos un par de algoritmos no es idéntico en términos del error de localización, y que además las diferencias no son causa del azar.

Tabla 5.6: Resultado del test de ANOVA para el escenario R4.

	Suma de cuadrados	gl	Media cuadrática	F	Sig
Entre-grupos	217299	4	54325	2,8962	0,02475
Intra-grupos	2344634	125	18757		

Tabla 5.7: Resultado del test de ANOVA para el escenario R3.

	Suma de cuadrados	gl	Media cuadrática	F	Sig
Entre-grupos	164339	4	41085	4,781	0,001271
Intra-grupos	1074174	125	8593		

Como el test de ANOVA concluyó que a lo menos uno de los promedios es distinto de otro, se procede a identificar cuáles de los algoritmos presenta dicha diferencia estadística, para esto se realiza una prueba de comparaciones múltiples (o pruebas post hoc) conocida por HSD de Tukey.

Tabla 5.8: Resultado de la prueba de Tukey para los escenarios R4 y R3.

Algoritmo	AGME-2	AGME-4	AGisla-2	AGisla-4
AGS	0,185	0,051	0,982	0,987
AGME-2		0,981	0,466	0,436
AGME-4			0,183	0,165
AGisla-2				0,999

(a) Escenario R4

Algoritmo	AGME-2	AGME-4	AGisla-2	AGisla-4
AGS	0,918	0,973	0,135	0,182
AGME-2		0,999	0,015*	0,023*
AGME-4			0,029*	0,043*
AGisla-2				0,999

(b) Escenario R3

Tabla 5.9: Medias obtenidas para el escenario R3.

Algoritmos	Medias
AGS	1217,135
AGME-2	1238,698
AGME-4	1232,782
AGisla-2	1156,648
AGisla-4	1160,23

En las tablas 5.8(a) y 5.8(b) se muestran los valores de significancia obtenidos por el test de comparaciones múltiples. Para el caso del escenario R3 (Tabla 5.8(b)) se observa

que se encuentran diferencias estadísticamente significativas en las medias para las combinaciones (AGME-2,AGisla-2), (AGME-2,AGisla-4), (AGME-4,AGisla-2) y (AGME-4,AGisla-4). Esto implica que los algoritmos con modelo de islas presentan mejores resultados promedios (ver tabla 5.9) que los modelos Maestro/Esclavo. Por otro lado, los valores de significancia obtenidos para el escenario R4 no son significativos (no existen diferencias entre los algoritmos), lo que lleva a una contradicción con el test de ANOVA con el cual se concluyó que a lo menos un par de algoritmos no son idénticos. Esto se debe a que el test de ANOVA es más sensible que el test de Tukey en la comparación de medias. Por lo que utilizando el test de Tukey no se puede identificar con una confianza suficiente en cual par de algoritmos existe una diferencia.

En la comparación de los resultados obtenidos en los escenarios R5 y R2, se hace uso del test no paramétrico Kruskal-Wallis, el cual compara las medianas para varios grupos independientes. Permite definir que las diferencias no se deben al azar (la diferencia estadística es significativa). En la Tabla 5.10 se define la hipótesis nula y alternativa para esta evaluación.

Tabla 5.10: Planteamiento de Hipótesis para el test de Kruskal-Wallis.

H_0	Los cinco algoritmos son idénticos en términos de la mediana del error de localización y las diferencias observadas son causa del azar.
H_1	Por lo menos un par de algoritmos no es idéntico en términos de la mediana del error de localización y las diferencias no son causa del azar.

Como se observa en la Tabla 5.11 los valores de significancia para los escenarios R5 y R2 no son significativos, lo cual implica que no existe evidencia estadística para rechazar la hipótesis nula. Esto significa que los algoritmos provienen de poblaciones idénticas y las diferencias observadas son causa del azar. Finalmente se puede concluir que este bloque de experimentos resulta poco relevante para la comparación final de los algoritmos en función de sus medianas.

Tabla 5.11: Resultados del test de Krukal-Wallis para los escenarios R5 y R2.

Escenario	Estadístico de K-W	Sig
R5	4,4777	0,3452
R2	6,5171	0,1637

Finalmente se concluye que sólo se obtiene diferencia significativa para los escenarios R4 y R3, se destacan los algoritmos AGisla-2 y AGisla-4. Esto implica que los resultados obtenidos por los distintos algoritmos provienen de poblaciones diferentes, lo cual significa que dependiendo del algoritmo que se utilice los resultados obtenidos pueden ser mejores o peores. Para los escenarios R5 y R2 no existe evidencia significativa, esto quiere decir que los algoritmos provienen de poblaciones idénticas, en otras palabras, independiente del algoritmo que se utilice los resultados obtenidos serán similares.

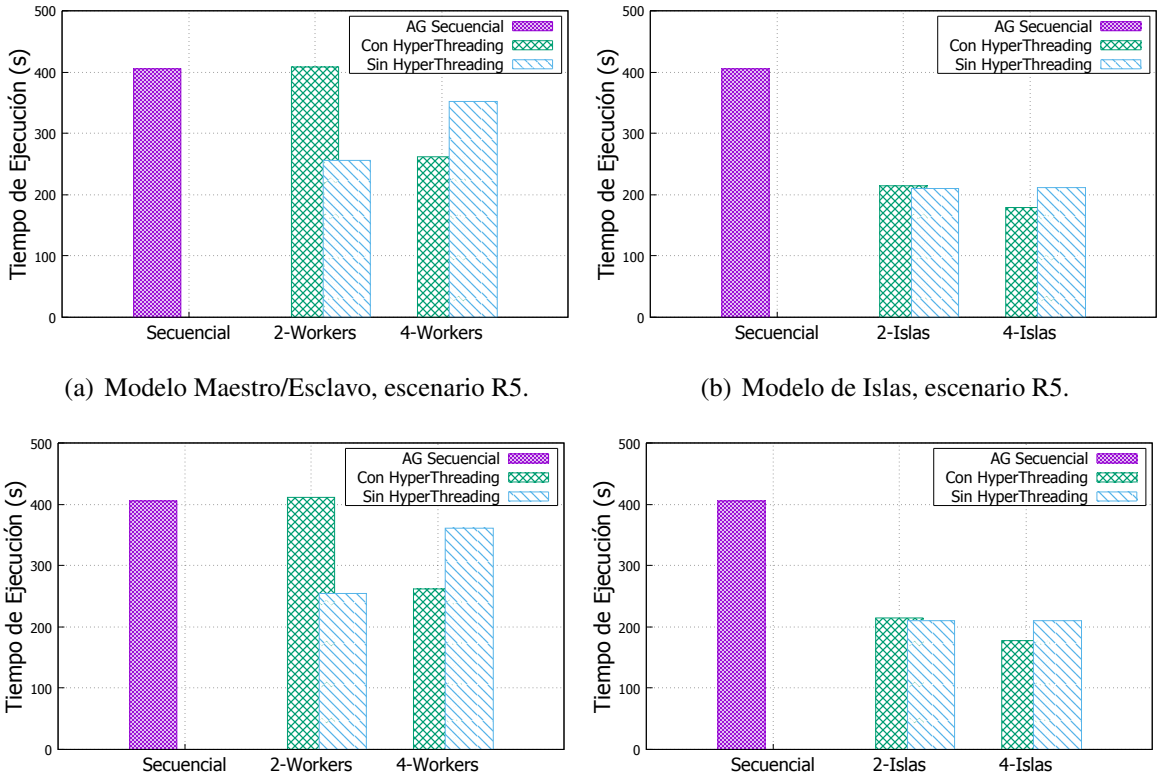
5.3.3 Evaluación del Tiempo de Procesamiento

Los tiempos de procesamiento necesarios por los tres sistemas estudiados (secuencial, 2-núcleos y 4-núcleos) para resolver los cuatro escenarios propuestos se presentan en la Figura 5.4. Como se puede observar, los algoritmos presentan tiempos de ejecución similares en los cuatro escenarios (ya que el criterio de parada es el mismo para todos los escenarios). También se aprecia la disminución del tiempo al aumentar el número de núcleos utilizados, aunque el tiempo de mejora de pasar de una arquitectura 2-núcleo (con hyperthreading desactivado) a una de 4-núcleo (hyperthreading activado) es mínima para ambos modelos paralelos. Esto principalmente se debe al diseño del computador (dos núcleos físicos y cuatro virtuales).

Como se puede apreciar en las Figuras 5.4(a)(c)5.5(a)(c), el modelo maestro/esclavo con dos trabajadores (también conocidos como workers) sobre una arquitectura sin hyperthreading es la que muestra el mejor tiempo de procesamiento. Por otro lado, para el modelo con cuatro trabajadores la opción con hyperthreading tiene un mejor rendimiento. En las Figuras 5.4(b)(d)5.5(b)(d) se observa la ejecución del algoritmo modelo de islas sobre una

arquitectura sin hyperthreading, el cual tiene igual comportamiento para dos y cuatro islas, pero para la ejecución sobre una arquitectura con hyperthreading se logra apreciar que el modelo con cuatro islas tiene una mejora en el tiempo de procesamiento.

Finalmente, se puede apreciar que en los cuatro escenarios el modelo de islas corriendo sobre una arquitectura de cuatro núcleos virtuales es el que presenta los mejores tiempos, obteniendo una mejora de a lo menos un 50% del tiempo promedio utilizado por un algoritmo genético secuencial y de un 30% con respecto al mejor algoritmo maestro/esclavo. Por último, en todos los escenarios el mejor tiempo siempre es obtenido por este algoritmo.



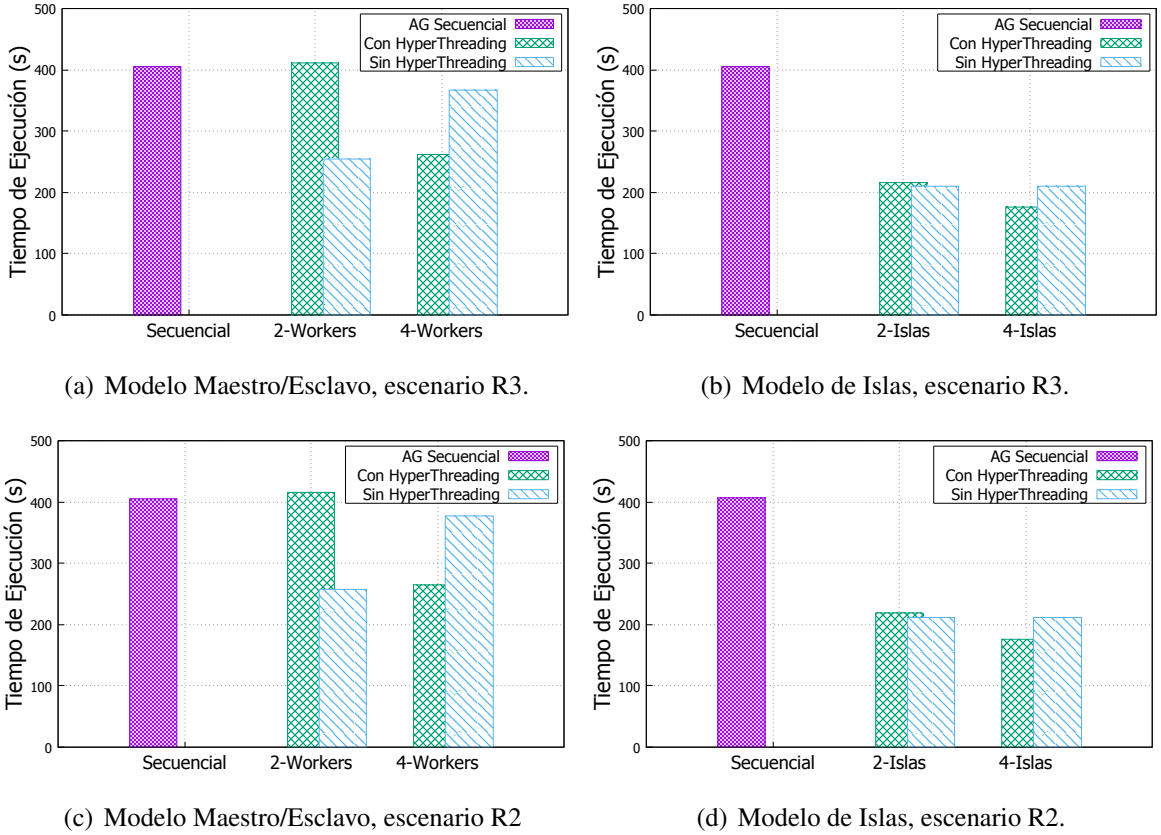
(a) Modelo Maestro/Esclavo, escenario R5.

(b) Modelo de Islas, escenario R5.

(c) Modelo Maestro/Esclavo, escenario R4.

(d) Modelo de Islas, escenario R4.

Figura 5.4 Tiempos de ejecución promedio para los escenarios R5 y R4.



(a) Modelo Maestro/Esclavo, escenario R3.

(b) Modelo de Islas, escenario R3.

(c) Modelo Maestro/Esclavo, escenario R2

(d) Modelo de Islas, escenario R2.

Figura 5.5 Tiempos de ejecución promedio para los escenarios R3 y R2.

5.3.4 Métricas de Desempeño

Los resultados mostrados en la Tabla 5.12 indican los valores medios de tiempo, *Speedup* y *eficiencia* conseguidos en las 26 ejecuciones independientes realizadas para cada instancia. Analizando los datos presentados en la Tabla 5.12, se puede concluir que ambos algoritmos paralelos presentan resultados aceptables. Si en primer lugar se analizan los resultados obtenidos por los algoritmos Maestro/Esclavo y modelo de islas sobre una arquitectura con hyperthreading desactivado, se puede observar que el algoritmo Maestro/Esclavo tiene un menor rendimiento en comparación con el modelo de islas el cual consigue valores cercanos al Speedup ideal, esto se debe al hecho de que el algoritmo Maestro/Esclavo consume más tiempo en la comunicación y, además, la condición de parada se debe cumplir de forma se-

cuencial, a diferencia del modelo de islas que se puede dividir en función de la cantidad de islas. Por ejemplo, una condición de parada de 30.000 generaciones en un modelo Maestro/Esclavo, será diferente en un modelo de isla, ya que el total de generaciones se divide en función de la cantidad de islas, por lo que se obtiene un condición de parada de 15.000 generaciones para cada isla.

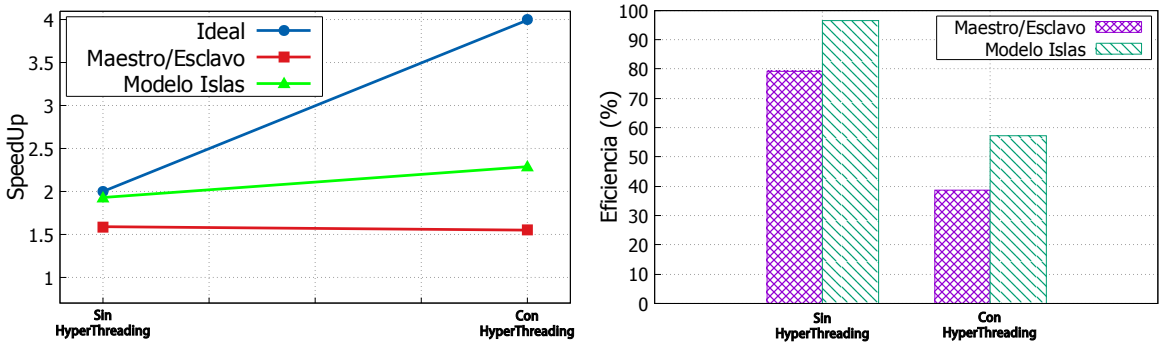
Tabla 5.12: Rendimiento de los algoritmos ($\overline{X}$ representa el tiempo medio en segundos, S_p es el *Speedup* y E_p es la *eficiencia*, donde p es la cantidad de núcleos).

	Secuencial	Sin Hyperthreading				Con Hyperthreading			
	AG Secuencial	Maestro/Esclavo		Modelo Islas		Maestro/Esclavo		Modelo Islas	
Instancias	$\overline{X}$	S_2	E_2	S_2	E_2	S_4	E_4	S_4	E_4
Escenario R5	405,56	1,59	79,27%	1,93	96,38%	1,55	38,78%	2,26	56,55%
Escenario R4	405,82	1,59	79,52%	1,93	96,62%	1,55	38,72%	2,28	57,00%
Escenario R3	404,93	1,59	79,43%	1,93	96,36%	1,54	38,61%	2,29	57,34%
Escenario R2	407,18	1,58	78,96%	1,92	96,23%	1,54	38,41%	2,31	57,75%
Media	405,88	1,59	79,29%	1,93	96,40%	1,55	38.63%	2,29	57,16%

Cuando los algoritmos se ejecutan con hyperthreading habilitado se utilizan los cuatro núcleos presentes en la arquitectura, con ésto se logra observar una baja en el Speedup y Eficiencia para el modelo de islas, lo cual es contradictorio, ya que se esperaría un aumento en ambos valores. Principalmente, esto se puede deber al hecho de que la arquitectura en la cual fueron realizados los experimentos, realmente no cuenta con cuatro núcleos físicos, como se describió en la sección 5.1, sino que solo con cuatro núcleos virtuales. Aunque en los tiempos de ejecución se obtienen mejoras (Figura 5.6(a)) no son las esperadas.

En la Figura 5.6(a) se muestra el speedup y eficiencia media conseguidas con los algoritmos paralelos. Para la primera gráfica se puede apreciar que el algoritmo modelo de islas, sí se ejecuta sin hyperthreading consigue un speedup cercano al ideal (speedup lineal). Cuando se habilita hyperthreading la eficiencia del modelo de islas disminuye, y el speedup se encuentra en el área de speedup sublineal. Por otro lado el modelo Maestro/Esclavo presenta un comportamiento lineal al pasar de la ejecución sin hyperthreading a la con hyperthread-

ing habilitado. Finalmente, el modelo Maestro/Esclavo se encuentra en el área del speedup sublienal.



(a) SpeedUp promedio encontrado.

(b) Eficiencia promedio encontrado.

Figura 5.6 Speedup y eficiencia media obtenida por los algoritmos paralelos.

Finalmente como se concluye que los algoritmos propuestos se encuentran afectados por la arquitectura del computador en el cual se realizaron los experimentos, para futuras implementaciones se recomienda una arquitectura con un mínimo de ocho núcleos.

5.3.5 Resultados para los Escenarios Propuestos

En esta sección se presentan los principales resultados obtenidos al aplicar los distintos algoritmos propuestos.

Para cada escenario de pruebas se presentan de forma gráfica las topologías obtenidas por cada algoritmo, reflejando el error de localización obtenido. Luego se evalúa el comportamiento del fitness para los algoritmos modelos de islas, para luego proceder a realizar una comparación del mejor fitness y el fitness promedio obtenido por todos los algoritmos.

Por cada escenario se presentan figuras con la comparación entre la topología obtenida y la real, donde los círculos verdes representan a los nodos anclas, los círculos azules representan la posición real de los nodos y las líneas rojas representan el error de localización entre la

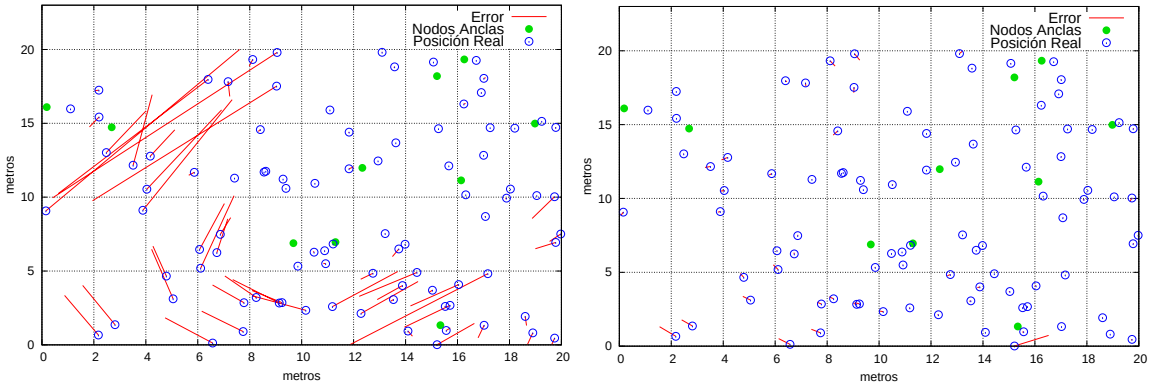
posición estimada y la posición real. El tamaño de las líneas representan que tan grande es el error o el fitness encontrado para cada individuo.

5.3.5.1 Pruebas para el Escenario R5

En las Figuras 5.7, 5.8 y 5.9 se muestran los resultados obtenidos por los algoritmos evolutivos propuestos.

De todas las topologías encontradas se logra apreciar que la obtenida por el algoritmo AGME-2 (Figura 5.7(b)) es la que cuenta con un menor error de localización total, esto se puede observar en la figura por la menor cantidad de líneas rojas presentes en la gráfica.

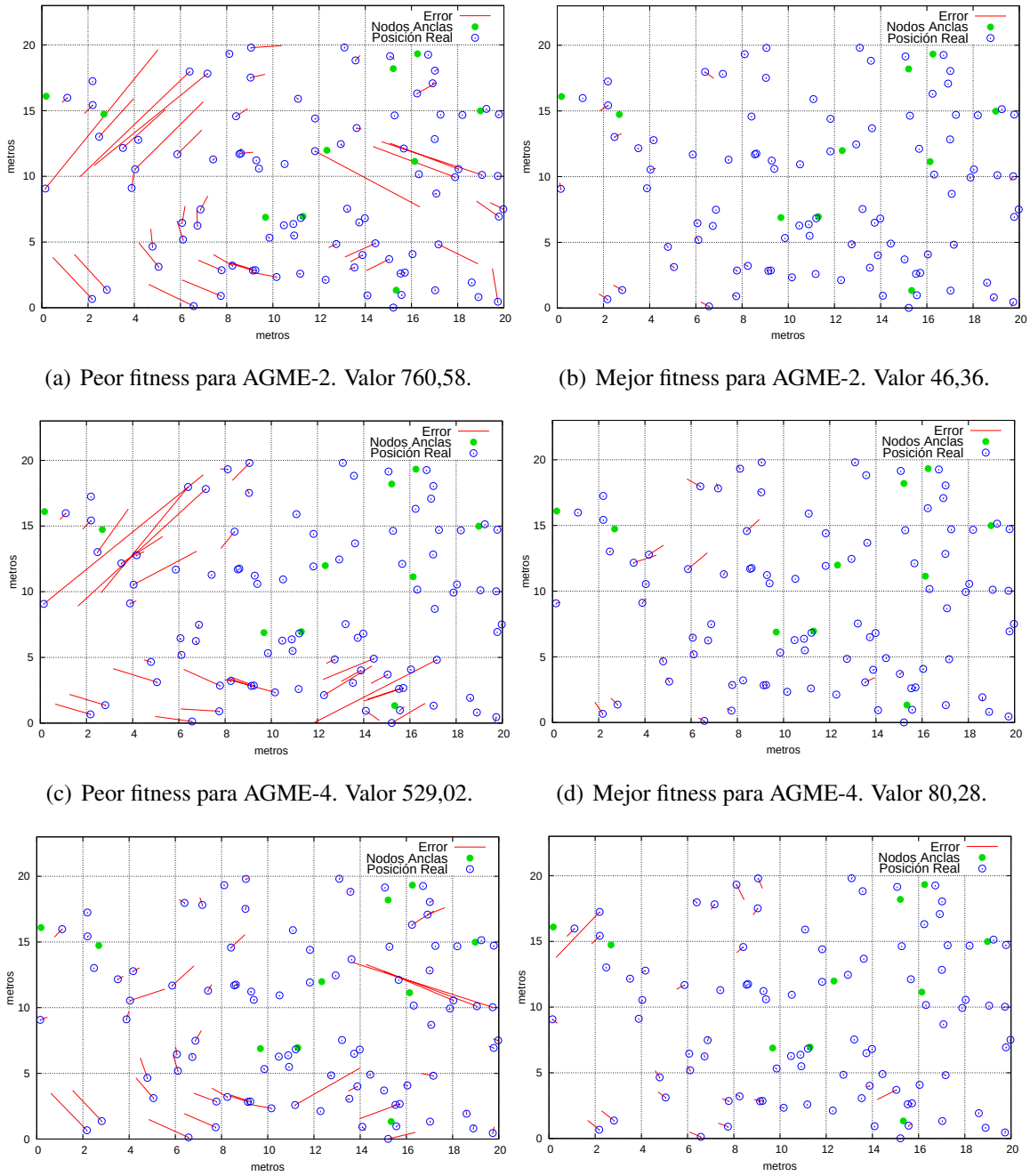
De los peores fitness encontrados por todos los algoritmos, los que presentan un menor error de localización acumulado son los modelos AGisla-2 y AGisla-4 (Figura 5.8(e)-(a)). Esto se complementa con las conclusiones obtenidas en las secciones anteriores, ya que estos algoritmos son los que presentan una menor diversidad en el total ejecuciones.



(a) Peor fitness para AGS. Valor 795,36.

(b) Mejor fitness para AGS. Valor 54,99.

Figura 5.7 Comparación entre la topología real vs la encontrada por el algoritmo AGS para el escenario R5.



(a) Peor fitness para AGME-2. Valor 760,58.

(b) Mejor fitness para AGME-2. Valor 46,36.

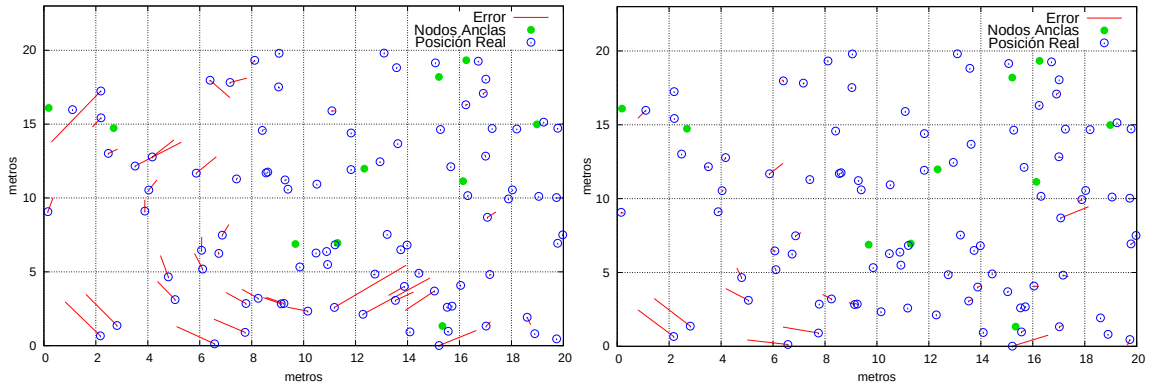
(c) Peor fitness para AGME-4. Valor 529,02.

(d) Mejor fitness para AGME-4. Valor 80,28.

(e) Peor fitness para AGisla-2. Valor 465,39.

(f) Mejor fitness para AGisla-2. Valor 99,09.

Figura 5.8 Comparación entre la topología real vs la encontrada por los algoritmos AGME-2, AGME-4 y AGisla-2 para el escenario R5.

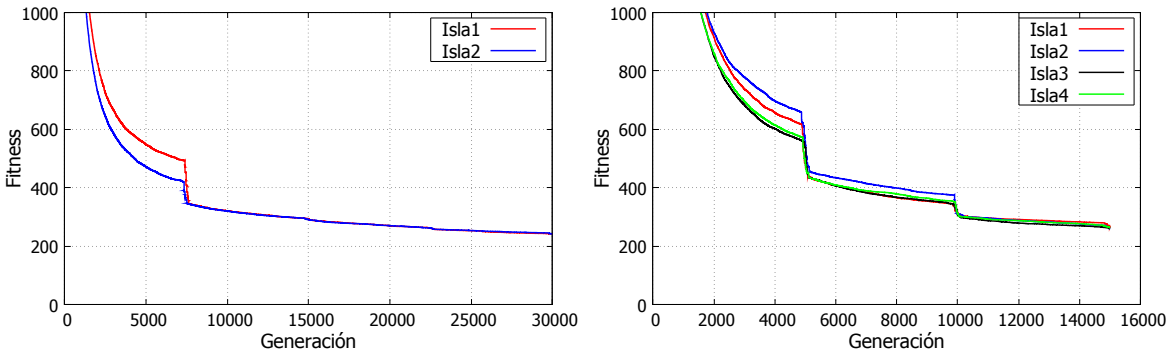


(a) Peor fitness para AGisla-4. Valor 373,69.

(b) Mejor fitness para AGisla-4. Valor 116,94.

Figura 5.9 Comparación entre la topología real vs la encontrada por el algoritmo AGisla-4 para el escenario R5.

En la Figura 5.10 se presenta el comportamiento del fitness promedio encontrado en las 26 ejecuciones independientes por los algoritmos AGisla-2 y AGisla-4. En la Figura 5.10(a) se puede observar que en el modelo de dos islas se producen los intercambios de individuos solo en la generación 7.500, también debería ocurrir en las generaciones 15000 y 22500, esto no se logra apreciar por el hecho de que las islas convergen de forma prematura en el primer intercambio. Para el modelo de cuatro islas (Figura 5.10(b)) el intercambio de material genético se produce en las generaciones 5.000 y 10.000. Por último, se logra apreciar que el modelo de 4 islas cuenta con una mejor convergencia en comparación al modelo de 2 islas, es decir, que necesita una menor cantidad de generaciones para encontrar una solución.



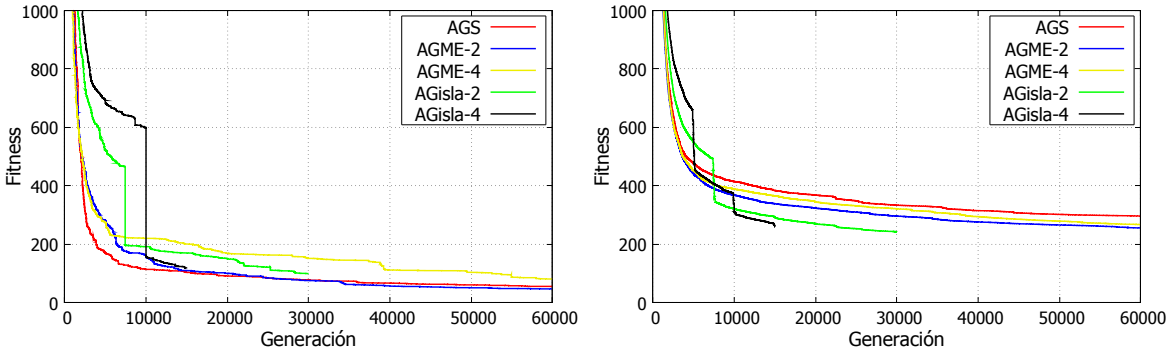
(a) Fitness para AGisla-2

(b) Fitness para AGisla-4

Figura 5.10 Comparación del fitness promedio obtenidos por los modelos AGisla-2 y AGisla-4 para el escenario R5.

Los gráficos de la Figura 5.11 muestran el comportamiento del mejor fitness encontrado en las 26 ejecuciones independientes y el fitness promedio. En el gráfico 5.11(a) se observa que los algoritmos AGS y AGME-2 son los que obtienen los mejores resultados y los que convergen más rápido. Además, se observa que los algoritmos modelos de islas comienzan con una convergencia más lenta que el resto, pero cuentan con cambios abruptos debido a las migraciones e inmigraciones de los individuos logrando obtener valores de fitness mejores que el modelo AGME-4 y cercanos a los modelos AGS y AGME-2, por otro lado existe la posibilidad que al aumentar la cantidad de generaciones a cada islas se obtengan resultados de mejor calidad.

En cuanto a la calidad promedio de los resultados, se puede apreciar (Figura 5.11(b)) que los algoritmos modelos de islas obtienen los mejores resultados, seguidos por el algoritmo AGME-2. Esta mejora en el promedio de los algoritmos de islas, como se explicó en secciones anteriores, se debe a la baja diversidad del fitness obtenido en las 26 ejecuciones independientes.



(a) Mejor fitness

(b) Fitness promedio

Figura 5.11 Comportamiento del mejor fitness y del fitness promedio para el escenario R5.

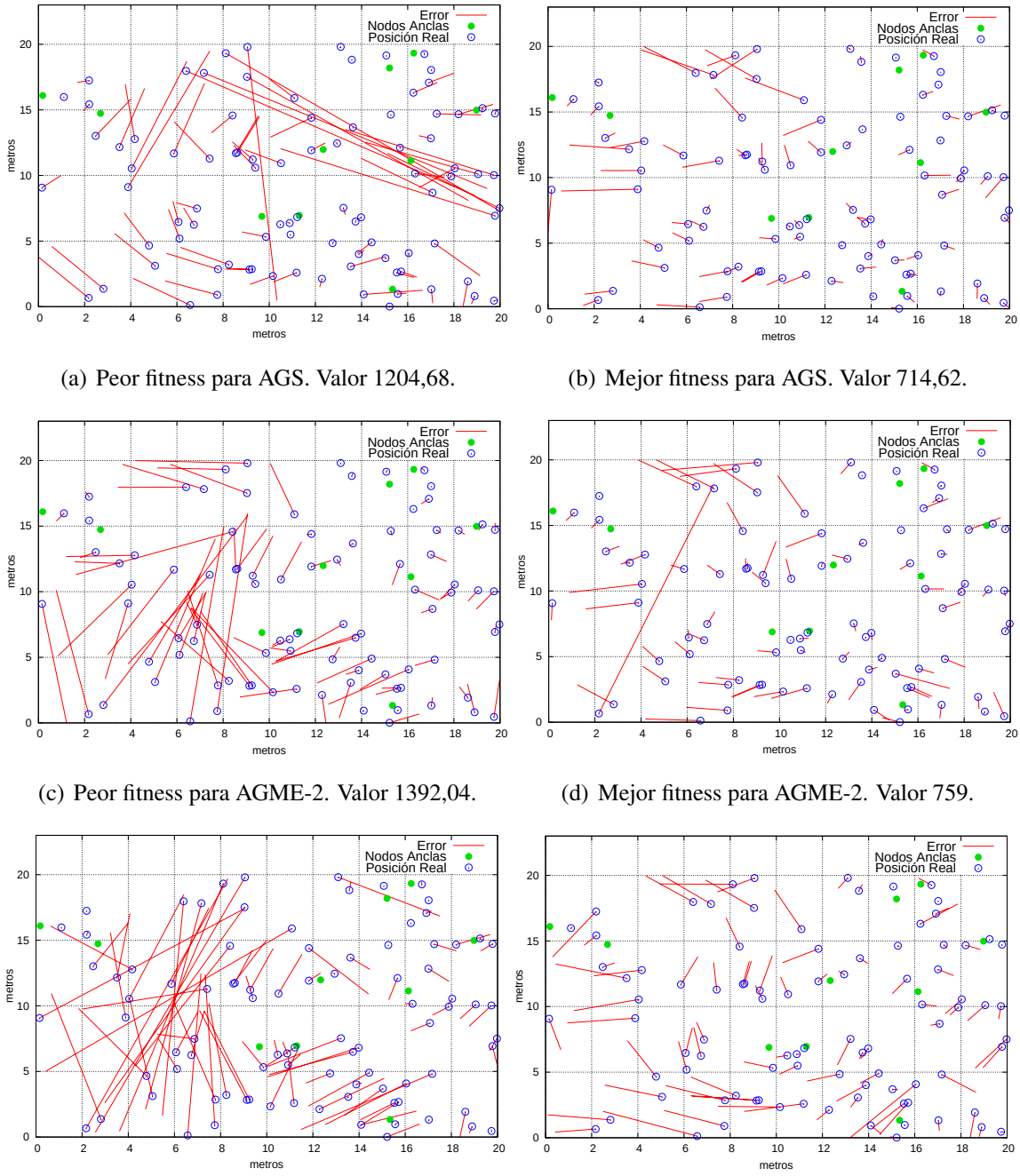
La forma que tiene el gráfico de convergencia en esta prueba será similar a la que se obtenga en las siguientes pruebas, puesto que no se modificarán los parámetros de los algoritmos y los escenarios son de igual tamaño. Sólo se espera obtener variaciones en los rangos del valor de fitness.

5.3.5.2 Pruebas para el Escenario R4

En las Figuras 5.12 y 5.13 se muestran los resultados obtenidos por los algoritmos. Estas figuras son el resultado de la comparación entre la topología obtenida y la real.

De todas las topologías encontradas se logra apreciar que la obtenida por el algoritmo AGisla-2 (Figura 5.12(d)) es la que cuenta con un menor error de localización, esto se observa por la menor cantidad de líneas rojas presentes en la gráfica. Por otro lado, los mayores errores de localización se producen en los sectores que no cuentan con a lo menos un nodo ancla.

A diferencia con el escenario anterior las peores topologías encontradas en las 26 ejecuciones independientes por los algoritmos, cuentan con un error de localización grande, sin destacar una sobre otra.



(a) Peor fitness para AGS. Valor 1204,68.

(b) Mejor fitness para AGS. Valor 714,62.

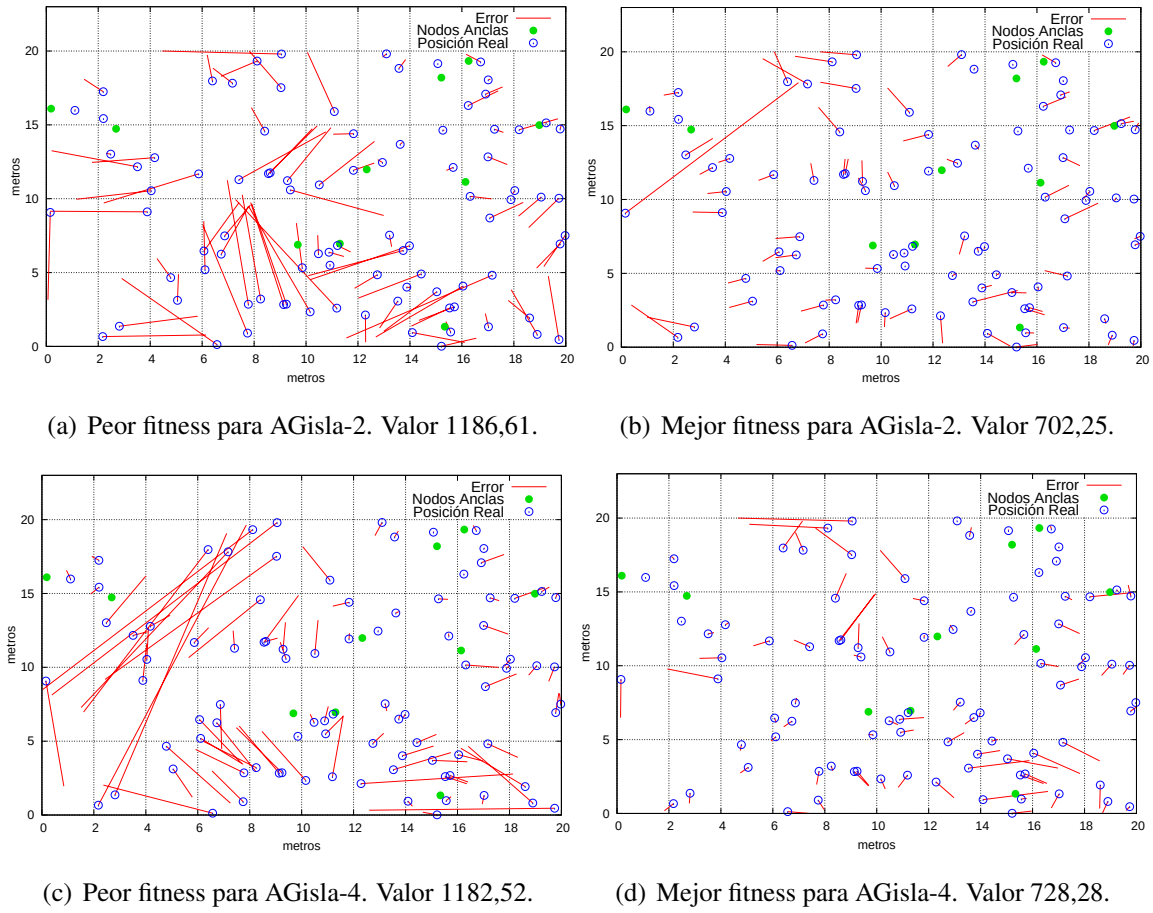
(c) Peor fitness para AGME-2. Valor 1392,04.

(d) Mejor fitness para AGME-2. Valor 759.

(e) Peor fitness para AGME-4. Valor 1334,02.

(f) Mejor fitness para AGME-4. Valor 790,51.

Figura 5.12 Comparación entre la topología real vs la encontrada por los algoritmos AGS, AGME-2 y AGME-4 para el escenario R4.



(a) Peor fitness para AGisla-2. Valor 1186,61.

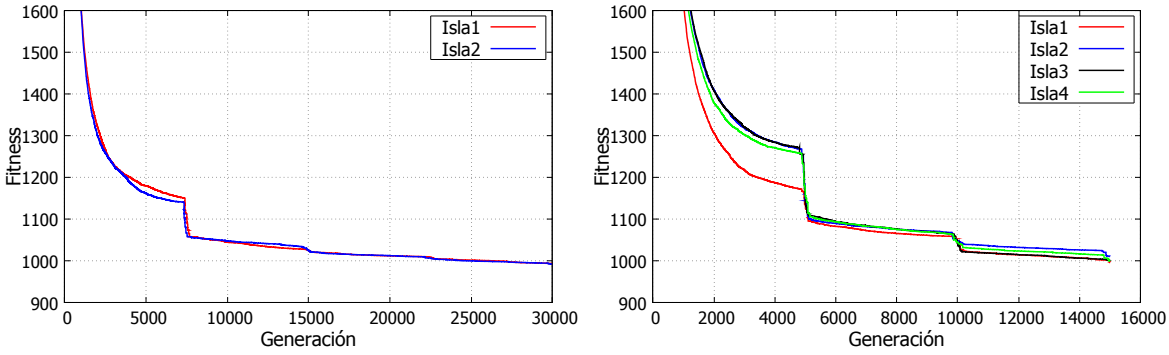
(b) Mejor fitness para AGisla-2. Valor 702,25.

(c) Peor fitness para AGisla-4. Valor 1182,52.

(d) Mejor fitness para AGisla-4. Valor 728,28.

Figura 5.13 Comparación entre la topología real vs la encontrada por los algoritmos AGisla-2 y AGisla-4 para el escenario R4.

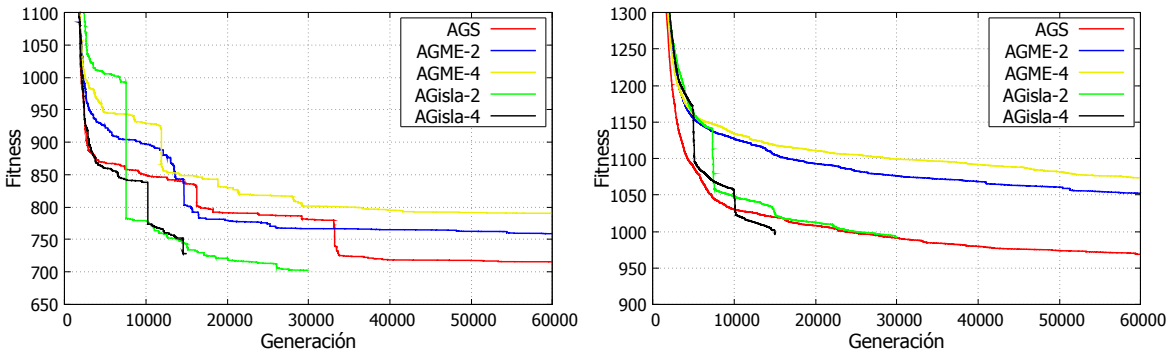
En la Figura 5.14 se presenta el comportamiento del fitness promedio. Como se comentó, la forma en que convergen los algoritmos es similar al escenario anterior, sólo cambian los rangos del fitness. De igual forma para el modelo de dos islas (Figura 5.14(a)) la migración e inmigración se producen en las generaciones 7.500, 15.000 y 22.500. Para el caso del modelo de cuatro islas se producen en las generaciones 5.000 y 10.000. Por último, se logra apreciar que ambos modelos presentan un comportamiento similar en la convergencia alcanzada, pero destacando que el AGisla-4 necesita la mitad de generaciones para alcanzarla.



(a) Fitness para AGisla-2 (b) Fitness para AGisla-4

Figura 5.14 Comparación del fitness promedio obtenidos por los modelos AGisla-2 y AGisla-4 para el escenario R4.

Los gráficos de la Figura 5.15 muestran el comportamiento del mejor fitness encontrado y el fitness promedio. En el gráfico 5.15(a) se observa que el algoritmo AGisla-2 es el que obtiene los mejores resultados y el que converge más rápido.



(a) Mejor fitness (b) Fitness promedio

Figura 5.15 Comportamiento del mejor fitness y del fitness promedio para el escenario R4.

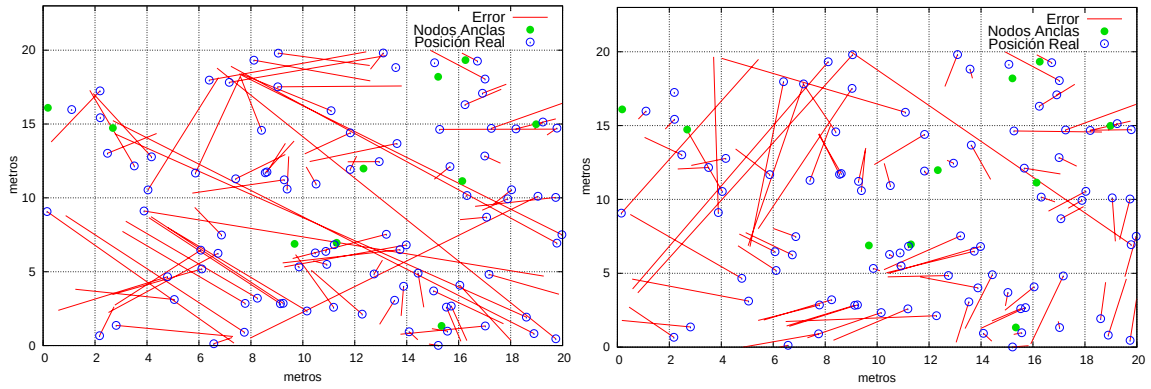
En cuanto a la calidad promedio de los resultados, se puede apreciar (Figura 5.11(b)) que los algoritmos modelos de islas y AGS son los que obtienen los mejores resultados. Es de

esperar que aumentando el criterio de parada en los algoritmos modelo de islas su valor de fitness promedio mejore.

5.3.5.3 Pruebas para el Escenario R3

En las Figuras 5.17 y 5.18 se muestran los resultados obtenidos. Se logra apreciar que conforme disminuye el radio de comunicación y el número de enlaces los resultados obtenidos por los algoritmos empeoran, obteniendo topologías con mucho error.

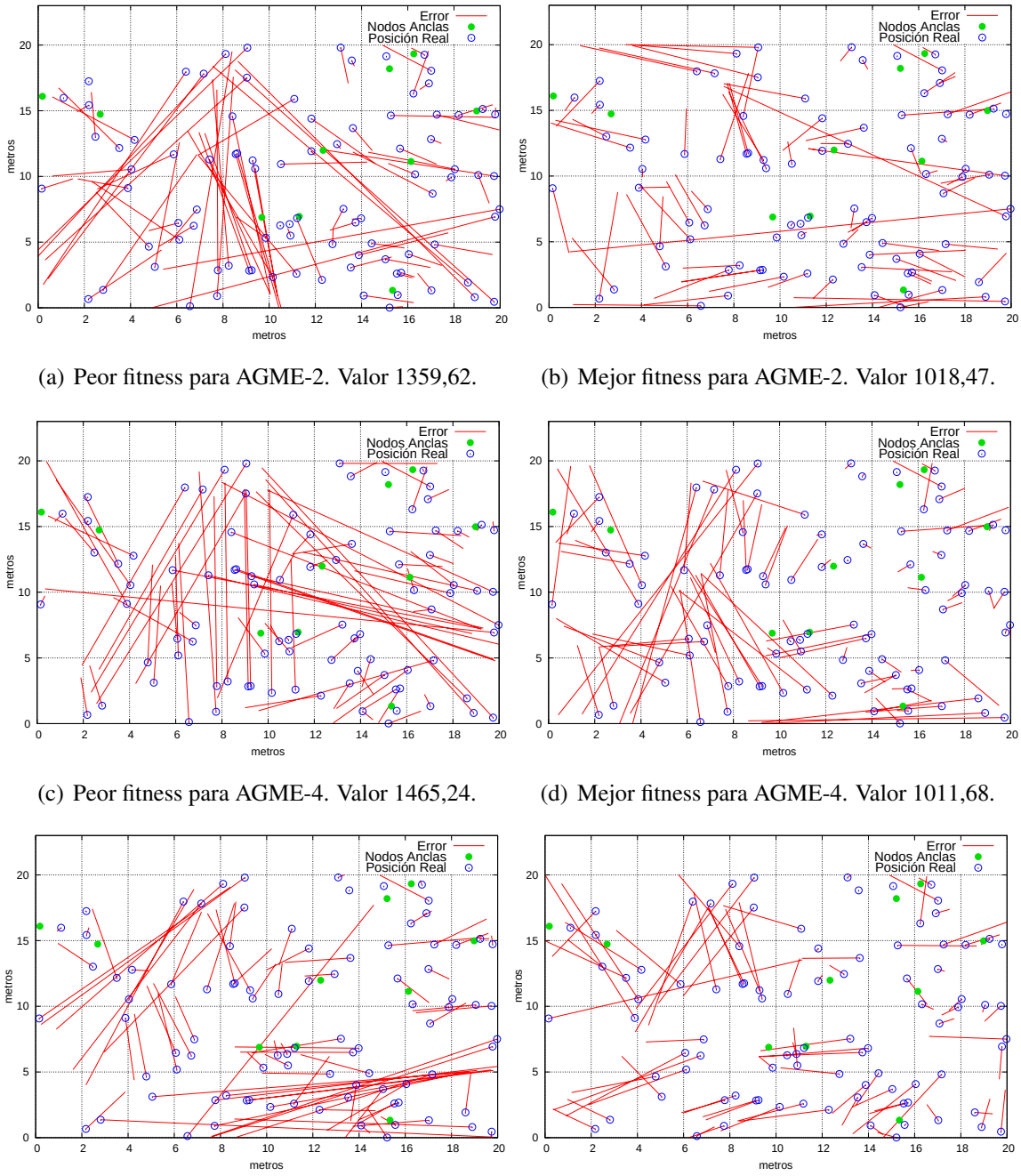
Finalmente de todas las topologías encontradas la que mejores resultados presenta es la obtenido por el algoritmo AGisla-2 (Figura 5.7(f)).



(a) Peor fitness para AGS. Valor 1399,13.

(b) Mejor fitness para AGS. Valor 1057,72.

Figura 5.16 Comparación entre la topología real vs la estimada por el algoritmo AGS para el escenario R3.



(a) Peor fitness para AGME-2. Valor 1359,62.

(b) Mejor fitness para AGME-2. Valor 1018,47.

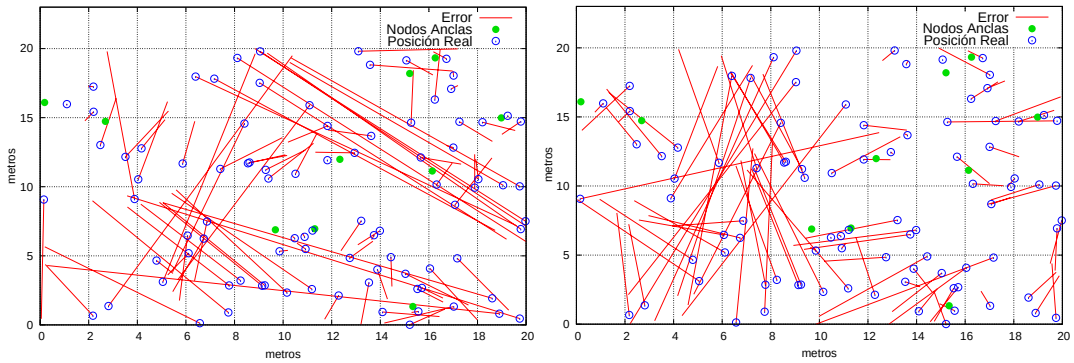
(c) Peor fitness para AGME-4. Valor 1465,24.

(d) Mejor fitness para AGME-4. Valor 1011,68.

(e) Peor fitness para AGisla-2. Valor 1299,47.

(f) Mejor fitness para AGisla-2. Valor 981,93

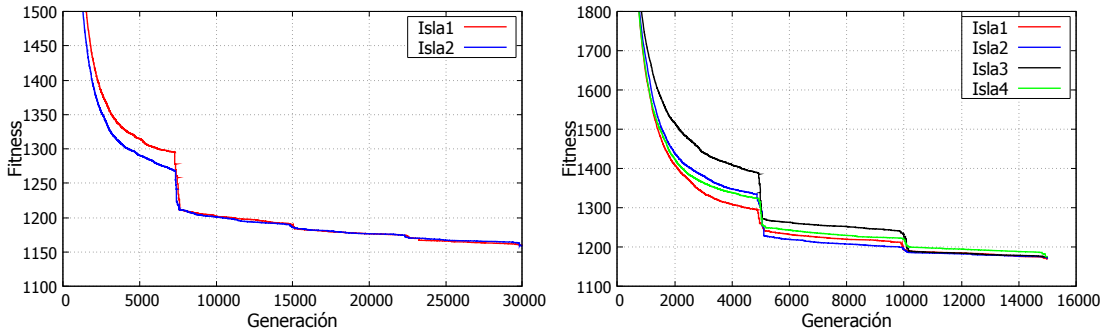
Figura 5.17 Comparación entre la topología real vs la encontrada por los algoritmos AGME-2, AGME-4 y AGisla-2 para el escenario R3.



(a) Peor fitness para AGisla-4. Valor 1317,42. (b) Mejor fitness para AGisla-4. Valor 1007,47.

Figura 5.18 Comparación entre la topología real vs la estimada por el algoritmo AGisla-4 para el escenario R3.

En la Figura 5.19 se presenta el comportamiento del fitness promedio. Como se comentó anteriormente, la forma en que convergen los algoritmos es similar a los escenarios anteriores, solo cambian los rangos del fitness. Por último, se logra apreciar que por una pequeña diferencia el modelo de dos islas obtiene el mejor resultado, pero se debe destacar que el modelo de cuatro islas necesita la mitad de generaciones.



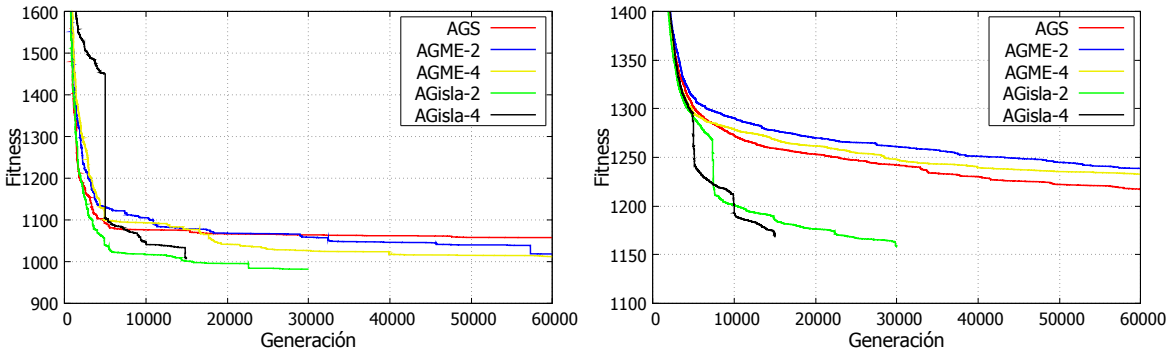
(a) Fitness para AGisla-2 (b) Fitness para AGisla-4

Figura 5.19 Fitness promedio obtenido por los modelos AGisla-2 y AGisla-4 para el escenario R3.

En el gráfico 5.20(a) se observa que el algoritmo AGisla-2 es el que obtiene los mejores

resultado y el que converge más rápido.

En cuanto a la calidad promedio de los resultados, se puede apreciar (Figura 5.20(b)) que los algoritmos modelos de islas son los que obtienen los mejores resultados.



(a) Mejor fitness

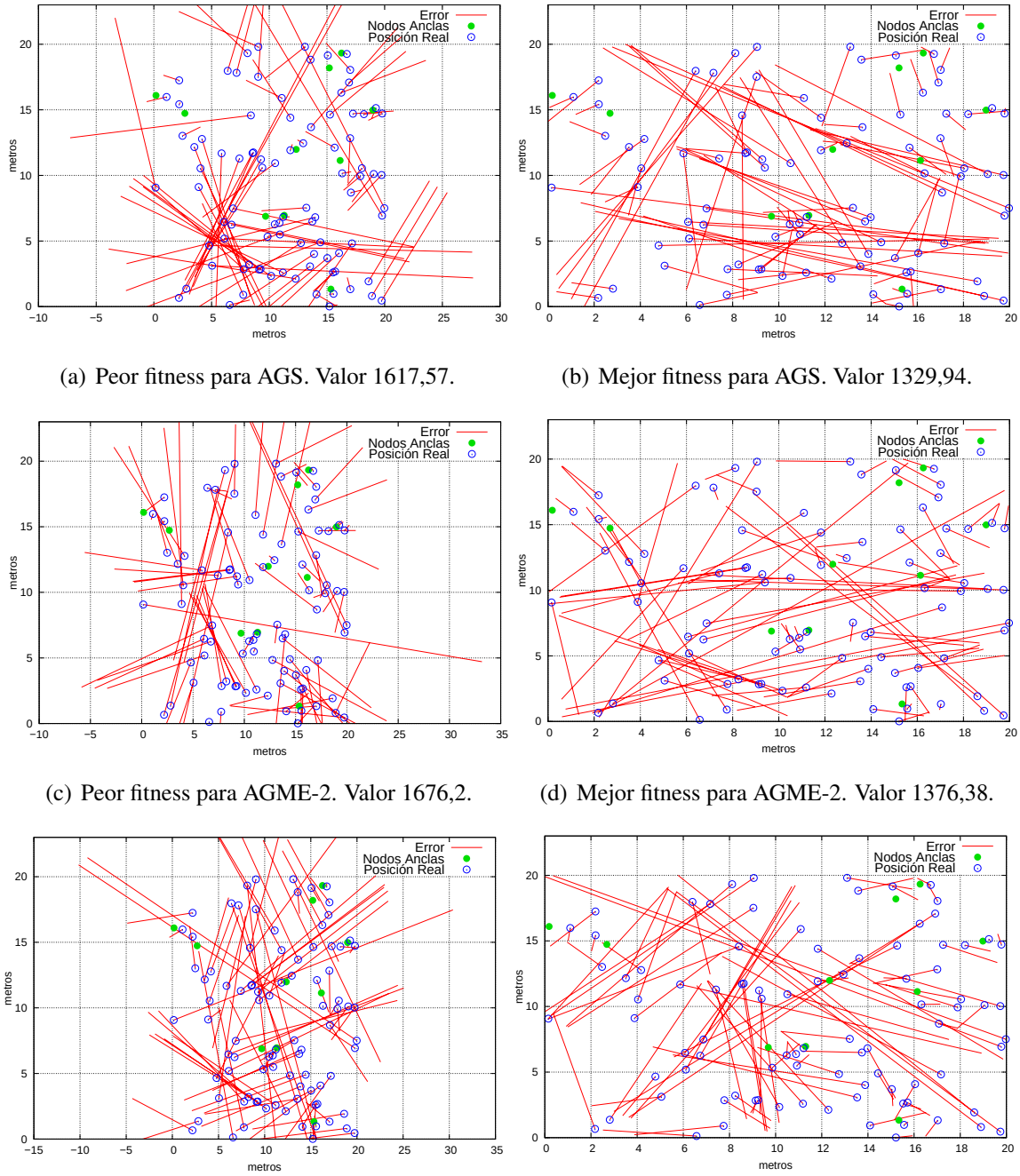
(b) Fitness promedio

Figura 5.20 Comportamiento del mejor fitness y del fitness promedio para el escenario R3.

5.3.5.4 Pruebas para el Escenario R2

En las Figuras 5.21 y 5.22 se muestran los resultados obtenidos por los algoritmos evolutivos propuestos.

Este es el escenario en el cual se obtienen los peores resultados, esto principalmente se debe a la disminución del radio de comunicación. Al contar con un menor radio de comunicación los nodos anclas no alcanzan a cubrir toda el área, por lo que algunos nodos estiman su posición en función de nodos que desconocen su propia posición, esto provoca que el error de localización se vaya acumulando. Solo obtienen buenas estimaciones los nodos que tengan una distancia igual o menor a dos metros con respecto a un nodo ancla.



(a) Peor fitness para AGS. Valor 1617,57.

(b) Mejor fitness para AGS. Valor 1329,94.

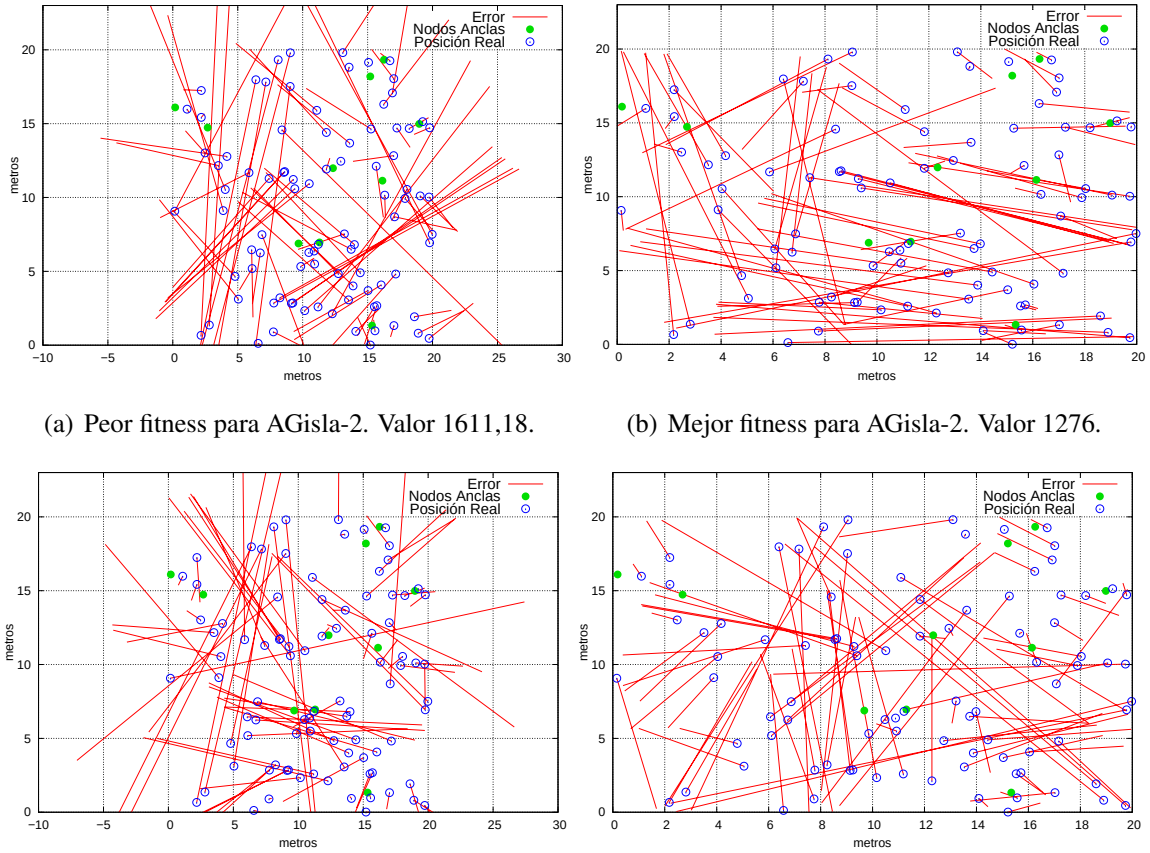
(c) Peor fitness para AGME-2. Valor 1676,2.

(d) Mejor fitness para AGME-2. Valor 1376,38.

(e) Peor fitness para AGME-4. Valor 1693,07.

(f) Mejor fitness para AGME-4. Valor 1377,62.

Figura 5.21 Comparación entre la topología real vs la encontrada por los algoritmos AGS, AGME-2 y AGME-4 para el escenario R2.



(a) Peor fitness para AGisla-2. Valor 1611,18.

(b) Mejor fitness para AGisla-2. Valor 1276.

(c) Peor fitness para AGisla-4. Valor 1734,37.

(d) Mejor fitness para AGisla-4. Valor 1346,16.

Figura 5.22 Comparación entre la topología real vs la encontrada por los algoritmos AGisla-2 y AGisla-4 para el escenario R2.

Finalmente, se descarta la comparación entre los fitness encontrado por los distintos algoritmos, ya que ningún de ellos es capaz de encontrar una solución aceptable.

CAPÍTULO 6

Conclusiones y Trabajo Futuro

6.1 Análisis de los objetivos planteados

Este trabajo de título tiene por objetivo general el diseño de un algoritmo genético paralelo para la resolución de problema de optimización combinatoria. Para ello, en primer lugar se realizó una revisión bibliográfica de los principales problemas NP-Complete que pueden ser encontrados en las Wireless Sensor Network (WSN), con el objetivo de seleccionar uno de estos problemas para la evaluación de los algoritmos propuestos. Finalmente se seleccionó el problema de localización de nodos.

Para la implementación de los algoritmos genéticos se utilizó el lenguaje de programación C++, por el hecho de ser un lenguaje que permite flexibilidad a la hora de programar. En la actualidad existen muchos frameworks que permiten disminuir el tiempo necesario en la implementación de las distintas metaheurísticas existentes, para discernir entre todas se realizó una exhaustiva revisión bibliográfica. La herramienta que mejor se adaptó a los requerimientos de este trabajo de título fue el framework ParadisEO, las características que destacan a este framework del resto son: se encuentra escrito en C++, cuenta con módulos para implementar distintos tipos de algoritmos genéticos, y además permite implementar otros tipos de metaheurísticas, lo cual habré las puertas para nuevas implementaciones.

Una vez definido el problema a resolver y las herramientas necesarias para la imple-

mentación de los algoritmos, se realizó una exhaustiva revisión bibliográfica que tuvo por objetivo aclarar los fundamentos básicos y las distintas etapas de los algoritmos genéticos, así como también describir los distintos tipos de algoritmos genéticos paralelos presentes en la literatura científica.

Se implementó un algoritmo genético secuencial (AGS), un algoritmo genético maestro/esclavo (AGME) y un algoritmo genético basado en un modelo de islas (AGislas). En los tres tipos de algoritmos se hizo uso de los mismo operadores genéticos. Los algoritmos fueron implementados sobre una arquitectura que cuenta con un procesador $Intel^{®}\ Core^{TM}$ i3, el cual cuenta con la tecnología hyperthreading. Este tipo de tecnología permite simular dos procesadores lógicos dentro de un único procesador físico. Al contar con dos procesadores físicos y la tecnología hyperthreading habilitada los resultados obtenidos para los algoritmos paralelos de cuatro núcleos no presentan el aumento de rendimiento esperado, logrando obtener un comportamiento de Speedup sublineal.

El comportamiento de cada modelo paralelo fue comparado con su equivalente secuencial, en función de los valores de fitness y tiempo de ejecución obtenidos. En la evaluación de los algoritmos se requirió una gran cantidad de tiempo, consumiendo un tiempo total de 60 horas para el total de ejecuciones.

Al comparar los resultados obtenidos por las técnicas implementadas, se observa que los algoritmos paralelos a excepción del AGME-2, presentan mejores tiempos de procesamiento en comparación al modelo secuencial, destacándose el algoritmos modelo de islas con hyperthreading habilitado, el cual obtiene a lo menos un 50% más de rendimiento en comparación al algoritmos AGS. En cuanto a los valores de fitness solo se obtiene diferencia significativa para los escenarios R4 y R3, lo cual implica que los resultados obtenidos por los respectivos algoritmos provienen de poblaciones diferentes, destacándose los algoritmos AGisla-2 y Agisla-4. Para los escenarios R5 y R2 no existe evidencia significativa, esto implica que los algoritmos provienen de poblaciones idénticas, en otras palabras, independiente del algoritmo que se utilice los resultados obtenidos serán similares.

Finalmente, los algoritmos modelos de islas son los que obtienen los mejores resultados en función del tiempo de ejecución. Respecto a los valores de fitness promedio los algoritmos AGS, AGisla-2 y AGisla-4 son los que obtienen los mejores resultados.

De los resultados obtenidos para los distintos escenarios propuestos, se puede concluir que conforme disminuye el radio de comunicación los nodos anclas no alcanzan a cubrir toda el área, por lo que algunos nodos no cuentan con nodos de referencia, y de esta forma deben estimar su posición en función de nodos que desconocen su propia posición, esto provoca que el error de localización se vaya acumulando. Para escenarios con una pequeña cantidad de enlaces de comunicación, solo obtendrán buenas estimaciones los nodos que tienen a lo menos un nodo ancla en su radio de comunicación.

En la etapa de implementación de los algoritmos, el principal objetivo era diseñar una herramienta que permitiera resolver cualquier escenario en el cual fuera necesario localizar los nodos. Si bien los resultados obtenidos para un escenario generado de forma aleatoria son prometedores, es necesario verificar su comportamiento frente a un problema real.

6.2 Trabajo futuro

Por la falta de tiempo y equipamiento fue necesario descartar algunas lineas de investigación de gran interés. Se sugieren algunos puntos para posteriores trabajos de investigación:

- Realizar una análisis estadístico que permita verificar la existencia de significancia estadística al variar los distintos operadores genéticos, operador de selección, operador de reemplazo y tamaño de la población.
- Para una mejor validación de los algoritmos propuestos se recomienda utilizar escenarios con distintos porcentajes de nodos anclas, además de utilizar escenarios con topologías tipo malla uniforme, forma de C, etc.
- Se propone implementar los algoritmos sobre un escenario real, para verificar su comportamiento en entornos reales.

- Implementar los algoritmos modelo maestro/esclavo y modelo de isla sobre una arquitectura que cuente con más de cuatro núcleos, con el objetivo de verificar con hasta que cantidad de núcleos se puede obtener un Speedup lineal.

Bibliografía

Alba, E. (1999). *Análisis y Diseño de Algoritmos Genético Paralelos Distribuidos*. PhD thesis, Universidad de Málaga.

Alcaraz, J., Maroto, C., & Ruiz, R. (2002). *Investigación operativa: modelos y técnicas de optimización*. Norwell, MA, USA: Universidad Politécnica de Valencia.

Arampatzis, T., Lygeros, J., & Manesis, S. (2005). A survey of applications of wireless sensors and wireless sensor networks. In *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*, (pp. 719–724).

Arce, A., Tech, A., Silva, A., & Costa, E. (2009). Monitorización de rebaños de bovinos a través de redes de sensores inalámbricos. *Arch. zootec*, *58*(222), 253–263.

Arora, A., Dutta, P., Bapat, S., Kulathumani, V., Zhang, H., Naik, V., Mittal, V., Cao, H., Gouda, M., Choi, Y., Herman, T., Kulkarni, S., Arumugam, U., Nesterenko, M., Vora, A., & Miyashita, M. (2004). A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks (Elsevier*, *46*, 605–634.

Boukerche, A., Oliveira, H., Nakamura, E., & Loureiro, A. (2007). Localization systems for wireless sensor networks. *Wireless Communications, IEEE*, *14*(6), 6–12.

Bruck, J., Gao, J., & Jiang, A. A. (2009). Localization and routing in sensor networks by local angle information. *ACM Trans. Sen. Netw.*, *5*(1), 7:1–7:31.

Cárcamo, H. & Bahamondes, J. (2014). Monitoreo ubicuo de salud en tiempo real con wbsn. *Ingeniare. Revista chilena de ingeniería*, *22*(2), 169–176.

Coello, C., Lamont, G., & van Veldhuizen, D. (2007). Mop evolutionary algorithm approaches. In *Evolutionary Algorithms for Solving Multi-Objective Problems*, Genetic and Evolutionary Computation Series (pp. 61–130). Springer US.

Czarn, A., MacNish, C., Vijayan, K., & Turlach, B. (2007). The detrimentality of crossover. In M. Orgun & J. Thornton (Eds.), *AI 2007: Advances in Artificial Intelligence*, volume 4830 of *Lecture Notes in Computer Science* (pp. 632–636). Springer Berlin Heidelberg.

Dargie, W. & Poellabauer, C. (2010). Localization. In J. Wiley & Sons (Eds.), *Funadamentals of Wireless Sensor Networks: Theory and Practice* (pp. 249–266). WILEY.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection*. New York. Appleton and Co.

Dasgupta, D. & Michalewicz, Z. (1997). *Evolutionary Algorithms in Engineering Applications*. U.S. Government Printing Office.

Davis, L. (1991). *Handbook of genetic algorithms*. VNR computer library. Van Nostrand Reinhold.

Deep, K. & Thakur, M. (2007). A new crossover operator for real coded genetic algorithms. *Applied Mathematics and Computation*, *188*(1), 895 – 911.

Eiben, A. E. & Smith, J. E. (2003). *Introduction to Evolutionary Computing*. SpringerVerlag.

Feng, J., Girod, L., & Potkonjak, M. (2006). Location discovery using data-driven statistical error modeling. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, (pp. 1–14).

Flynn, M. (1972). Some computer organizations and their effectiveness. *Computers, IEEE Transactions on*, *C-21*(9), 948–960.

Gagné, C. & Parizeau, M. (2006). Genericity in evolutionary computation software tools: Principles and case study. *International Journal on Artificial Intelligence Tools*, *15*(2), 173–194. 22 pages.

García, C., Ibargüengoytia, P., García, J., & Pérez, J. (2007). Wireless sensor networks and applications: a survey. *IJCSNS International Journal of Computer Science and Network Security*, *7*, 264–273.

García, S., Molina, D., Lozano, M., & Herrera, F. (2007). Un estudio experimental sobre el uso de test no paramétricos para analizar el comportamiento de los algoritmos evolutivos en problemas de optimización. *Congreso Español sobre Metaheurísticas*, *V*, 275 – 285.

Gen, M. & Cheng, R. (1996). A survey of penalty techniques in genetic algorithms. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, (pp. 804–809).

Goldberg, D. E. & robert (1985). Alleles, loci, and the traveling salesman problem. In Grefenstette, J. J. (Ed.), *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*. Lawrence Erlbaum Associates, Publishers.

González, M. (2011). *Soluciones Metaheurísticas al "Job-Shop Scheduling Problem with Sequence-Dependent Setup Times"*. PhD thesis, Universidad de Oviedo.

Harik, G. R., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, *7*(3), 231–253.

Haupt, R. L. & Haupt, S. E. (2004). *Practical Genetic Algorithms*. New York, NY, USA: John Wiley & Sons, Inc.

Hennessy, J. & Patterson, D. (1993). *Arquitectura de computadores: un enfoque cuantitativo*. McGraw-Hill.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: University of Michigan Press.

Hsiao-Lan, F. (1994). *Genetic Algorithms in Timetabling and Scheduling*. PhD thesis, University of Edinburgh.

HUANG, H., CAI, Q., WANG, R., SHE, B., & MA, Y. (2011). The wireless sensor network localization algorithm based on improved pattern genetic algorithm. *Journal of Computational Information Systems*, *7*(9), 3031–3038.

Janikow, C. Z. & Michalewicz, Z. (1991). An experimental comparison of binary and floating point representations in genetic algorithms. In Belew, R. K. & Booker, L. B. (Eds.), *ICGA*, (pp. 31–36). Morgan Kaufmann.

Jiang, N., Jin, S., Guo, Y., & He, Y. (2013). Localization of wireless sensor network based on genetic algorithm. *International Journal of Computers Communications & Control*, *8*(6), 825–837.

Johnson, E. E. (1988). Completing an mimd multiprocessor taxonomy. *SIGARCH Comput. Archit. News*, *16*(3), 44–47.

Kaya, M. (2011). The effects of two new crossover operators on genetic algorithm performance. *Applied Soft Computing*, *11*(1), 881–890.

Kim, Y.-J., Govindan, R., Karp, B., & Shenker, S. (2005). Geographic routing made practical. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, (pp. 217–230)., Berkeley, CA, USA. USENIX Association.

Larranaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., & Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artif. Intell. Rev.*, *13*(2), 129–170.

Lédeczi, A., Nádas, A., Völgyesi, P., Balogh, G., Kusy, B., Sallai, J., Pap, G., Dóra, S., Molnár, K., Maróti, M., & Simon, G. (2005). Countersniper system for urban warfare. *ACM Trans. Sen. Netw.*, *1*(2), 153–177.

Liao, Y.-H. & Sun, C.-T. (2001). An educational genetic algorithms learning tool. *Education, IEEE Transactions on*, *44*(2), 20 pp.–.

Lin, C.-H. (2013). A rough penalty genetic algorithm for constrained optimization. *Information Sciences*, *241*(0), 119 – 137.

Luque, G. & Alba, E. (2011). *Parallel Genetic Algorithms: Theory and Real World Applications*. Studies in Computational Intelligence. Springer.

Ma, Z. S. (2012). Chaotic populations in genetic algorithms. *Applied Soft Computing*, *12*(8), 2409 – 2424.

Ma, Z. S. & Krings, A. W. (2008). Dynamic populations in genetic algorithms. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, (pp. 1807–1811)., New York, NY, USA. ACM.

Maaranen, H., Miettinen, K., & Mäkelä, M. M. (2004). Quasi-random initial population for genetic algorithms. *Computers & Mathematics with Applications*, *47*(12), 1885 – 1895.

Malan, K. M. & Engelbrecht, A. P. (2013). A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, *241*(0), 148 – 163.

Michalewicz, Z. (1994). *Genetic Algorithms + Data Structures = Evolution Programs (2Nd, Extended Ed.)*. New York, NY, USA: Springer-Verlag New York, Inc.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press.

Molina, G. (2010). *Técnicas de optimización para redes de sensores*. PhD thesis, Universidad de Málaga.

Molina, G. & Alba, E. (2011). Location discovery in wireless sensor networks using metaheuristics. *Appl. Soft Comput.*, *11*(1), 1223–1240.

Moore, D., Leonard, J., Rus, D., & Teller, S. (2004). Robust distributed network localization with noisy range measurements. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, (pp. 50–61)., New York, NY, USA. ACM.

Moreno, C. (2011). Localización en redes inalámbricas de sensores. Master's thesis, Universidad Autónoma Metropolitana.

Nan, G. & Li, M. (2008). Evolutionary based approaches in wireless sensor networks: A survey. In *Natural Computation, 2008. ICNC '08. Fourth International Conference on*, volume 5, (pp. 217–222).

Niewiadomska, E., Marks, M., & Kamola, M. (2011). localization in wireless sensor networks using heuristic optimization techniques. *Journal of Telecommunications and information technology*, *4*(9), 55–64.

Nowostawski, M. & Poli, R. (1999). Parallel genetic algorithm taxonomy. In *Knowledge-Based Intelligent Information Engineering Systems, 1999. Third International Conference*, (pp. 88–92).

Parejo, J. A. (2013). *MOSES: A Metaheuristic Optimization Software ECOSYSTEM*. PhD thesis, Universidad de Sevilla.

Parejo, J. A., Ruiz-Cortés, A., Lozano, S., & Fernandez, P. (2012). Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing*, *16*(3), 527–561.

Parhami, B. (2007). *Arquitectura de Computadores de los microprocesadores a las supercomputadoras*. McGraw-Hill.

Pinedo, M. (2008). *Scheduling: theory, algorithms, and systems*. Prentice Hall international series in industrial and systems engineering. Prentice Hall.

Ramirez, E. (2010). Using genetic algorithms to solve high school course timetabling problems. Master's thesis, San Diego State University.

Ramos, E. (2004). Maximización de la disponibilidad en líneas de producción por medio de programación multiobjetivo. Master's thesis, Universidad de las Américas Puebla.

Rodríguez, P. & Risso, D. (2009). Computación paralela. Master's thesis, Universidad del Bío-Bío.

Romero, A., Marín, A., & Jiménez, J. (2013). Red de sensores inalámbricos para el monitoreo de alertas tempranas en minas subterráneas: una solución a la problemática de atmósferas explosivas en la minería de carbón en colombia. *Ingeniería y Desarrollo*, *31*, 227 – 250.

Salomon, R. (1996). Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. a survey of some theoretical and practical aspects of genetic algorithms. *Biosystems*, *39*(3), 263 – 278.

Salto, C. (2000). Algoritmos evolutivos avanzados como soporte del proceso productivo. Master's thesis, Universidad Nacional de la Plata.

Sivanandam, S. N. & Deepa, S. N. (2008). Terminologies and operators of GA. In *Introduction to Genetic Algorithms* (pp. 39–81). Springer Berlin Heidelberg.

Suárez, V. F., Guerrero, l., & Castrillón, O. D. (2013). Programación de horarios escolares basados en ritmos cognitivos usando un algoritmo genético de clasificación no-dominada, nsga-ii. *Información tecnológica*, *24*, 103 – 114.

Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*. Wiley Publishing.

Tam, V., yip Cheng, K., & shan Lui, K. (2006). Using micro-genetic algorithms to improve localization in wireless sensor networks. *J. of Comm., Academy Publisher*, *1*(4), 1–10.

Tanenbaum, A. S. (2012). *Structured Computer Organization.* (6th ed.). Pearson Education.

Thakur, M. (2014). A new genetic algorithm for global optimization of multimodal continuous functions. *Journal of Computational Science*, *5*(2), 298 – 311. Empowering Science through Computing + BioInspired Computing.

Thakur, M., Meghwani, S. S., & Jalota, H. (2014). A modified real coded genetic algorithm for constrained optimization. *Applied Mathematics and Computation*, *235*, 292–317.

WANG, X., SUN, Z., & JI, Z. (2013). Genetic algorithm for wireless sensor network localization with level-based reliability scheme. *Journal of Computational Information Systems*, *9*(16), 6479–6486.

Wright, A. H. (1991). Genetic algorithms for real parameter optimization. In *Foundations of Genetic Algorithms*, (pp. 205–218). Morgan Kaufmann.

Yick, J., Mukherjee, B., & Ghosal, D. (2008). Wireless sensor network survey. *Computer Networks*, *52*(12), 2292 – 2330.

Yoneki, E. & Bacon, J. (2005). A survey of wireless sensor network technologies: research trends and middleware's role. Technical Report UCAM-CL-TR-646, University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom, phone +44 1223 763500.

Zhang, P., Sadler, C. M., Lyon, S., & Martonosi, M. (2004). Hardware design experiences in zebranet. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, (pp. 227–238)., New York, NY, USA. ACM.

Zhang, Q., Wang, J., Jin, C., Ye, J., Ma, C., & Zhang, W. (2008). Genetic algorithm based wireless sensor network localization. In *Natural Computation, 2008. ICNC '08. Fourth International Conference on*, volume 1, (pp. 608–613).

Anexos

Anexo A

ParadisEO

A.1 Instalación en Windows

A.1.1 Herramientas Necesarias

Antes de comenzar se deben instalar las aplicaciones *CMake* y *MinGW*, en la pagina oficial del proyecto se puede encontrar un paquete que incluye estas herramientas.

CMake, es un gestor de archivos (define la secuencia), generación o automatización de código. Permite configurar proyectos para distintas plataformas (Windows, Linux, Mac) y compiladores (gcc, MSVC). Puede compilar código fuente, crear librerías, generar wrappers y construir ejecutables en distintas combinaciones.

El proceso de construcción es controlado por la creación de uno o más ficheros *CMakeLists.txt* por cada directorio (incluyendo subdirectorios) que forma un proyecto. Cada *CMakeLists.txt* consiste en uno o más comandos, estos comando tiene la siguiente forma *COMANDO(argumentos)*.

Por otra parte *MinGW*, es una adaptación del compilador gcc de Linux para Windows.

A.1.2 Instalación

Para la instalación del framework ParadisEO se debe descargar el archivo fuente *ParadisEO-2.0.1.zip*, que se encuentra en la página oficial. A continuación se describen los pasos recomendados para la instalación del código fuente:

1. Ejecutar *cmd* de Windows.
2. Extraer el código desde el archivo descargado.
3. Ingresar a la carpeta donde se encuentran los archivos extraídos.
4. Ejecutar la siguiente secuencia: *Usurario$ mkdir build && cd build*.
5. Dentro de la carpeta build se procede a descomprimir el código fuente con el siguiente comando: *Usuario$ cmake .. -G "MinGW Makefiles" -DINSTALL_TYPE=full*.
6. Finalmente se debe compilar el código de ParadisEO ejecutando *Usuario$ make*.

A.2 Instalación en Linux Debian

A.2.1 Herramientas Necesarias

Antes de comenzar se deben instalar las aplicaciones *CMake*, *gcc* y *g++*, en la pagina oficial del proyecto se puede encontrar un paquete que incluye estas herramientas.

CMake, es un gestor de archivos (define la secuencia), generación o automatización de código. Permite configurar proyectos para distintas plataformas (Windows, Linux, Mac) y compiladores (gcc, MSVC). Puede compilar código fuente, crear librerías, generar wrappers y construir ejecutables en distintas combinaciones.

El proceso de construcción es controlado por la creación de uno o más ficheros *CMakeLists.txt* por cada directorio (incluyendo subdirectorios) que forma un proyecto. Cada *CMakeLists.txt* consiste en uno o más comandos, estos comando tiene la siguiente forma *COMANDO(argumentos)*.

Para la instalación de *CMake* en Linux se puede seguir dos caminos, a continuación se describen las dos alternativas:

- Descargar, compilar y instalar desde código:
  1. Descargar el código fuente desde *www.cmake.org*.
  2. Extraer el código desde el archivo descargado.
  3. Configurar, *Usuario$ ./configure -prefix=/opt/cmake*. Define donde se instalara *CMake*.
  4. Compilación, *Usuario$ make*.
  5. Instalación, *Usuario$ make install*.
  6. Si después de la instalación no se presenta ningún error se puede verificar la instalación ejecutando el siguiente comando *Usuario$ /opt/cmake/bin/cmake -version*.
- Utilizar el administrador de paquetes:
  1. Simplemente se ejecuta el comando *Usuario$ apt-get install cmake*.

Principalmente las alternativas propuestas se diferencia por el motivo de que la primera permite instalar la última versión disponible de *CMake*, y la segunda sólo instala la versión estable disponible en los repositorios de Linux.

Por último *gcc* y *g++* ya se encuentran instalados en Linux Debian 7.5 con la versión 4.7.2, pero es necesario instalar la dependencia *build-essential* para que *CMake* reconozca los compiladores.

A.2.2 Instalación

Para la instalación del framework ParadisEO se debe descargar el archivo fuente *ParadisEO-2.0.1-tar.gz*, que se encuentra en la página oficial. Si se quiere obtener las ultimas modificaciones del modulo SMP que no se encuentran en el archivo anterior se debe descargar el framework desde la siguiente dirección https://gforge.inria.fr/git/paradiseo/paradiseo.git. A continuación se describen los pasos recomendados para la instalación del código fuente:

1. Ejecutar el *Terminal*.
2. Extraer el código desde el archivo descargado.
3. Ingresar a la carpeta donde se encuentran los archivos extraídos.
4. Ejecutar la siguiente secuencia: `Usuario$ mkdir build && cd build`. Crea e ingresa a la carpeta build.
5. Para una instalación completa que incluye ejemplos, lecciones y librerías, se debe utilizar el siguiente comando: `Usuario$ cmake .. -DINSTALL_TYPE=full`.
6. Opcionalmente se puede habilitar la instalación de los módulos SMP o PEO (solo es posible instalar uno), ver A.2.2.1 y A.2.2.2 respectivamente.
7. Finalmente se debe compilar el código de ParadisEO ejecutando `Usuario$ make`.

A.2.2.1 Modulo SMP

- Para la instalación del modulo SMP se debe contar como mínimo con *gcc-4.7* y *g++-4.7*.
- El modulo SMP se encuentra incluido en el archivo principal de ParadisEO.
- Finalmente para compilar el modulo SMP en la compilación de ParadisEO es necesario ejecutar el comando: `Usuario$ cmake .. -DSMP=true`.
- Opcionalmente si existen problemas en la compilación podría ser necesario especificar la ubicación de los compiladores de la siguiente forma: `Usuario$ cmake .. -DCMAKE_C_COMPILER=/usr/bin/gcc -DCMAKE_CXX_COMPILER=/usr/bin/g++`.
- Mientras se compila el framework no encontrara los paquetes *doxygen* y *lcov*, los cuales son opcionales.

A.2.2.2 Modulo PEO

- Para la instalación del modulo PEO se debe contar con las dependencias *libxml2*, *libxml2-dev*, y el paquete *mpich2*.
- El modulo PEO no está incluido en el archivo principal de ParadisEO, por lo tanto es necesario descarga el archivo *peo-2.0.1.tar.gz* desde la pagina.
- Luego se debe agregar a la carpeta principal de ParadisEO los archivos extraídos.
- Finalmente para compilar el modulo PEO en la compilación de ParadisEO es necesario ejecutar el comando: *Usuario$ cmake .. -DPEO=true*.

A.3 Generando el primer proyecto

1. Primero se debe descargar el archivo HowToDoContribution.zip o HowToDoContribution.tar.gz desde la pagina de ParadisEO.
2. Una vez descomprimido se ingresa a la carpeta MyContrib, dentro de esta carpeta se presentan los siguientes archivos:
   - application, en esta carpeta se debe agregar el código principal, es obligatoria.
   - Build, es la carpeta donde se compilara el algoritmo.
   - Doc, en esta carpeta se agregan la documentación para el algoritmo, no es obligatoria.
   - src, en esta carpeta se agregan los codigos diseñados, no es obligatoria.
   - test, en esta carpeta se pueden agregar test para verificar si el código realizado se ejecuta como uno desea, no es obligatoria.
   - CMakeLists, es el archivo que indica toda la información necesaria para generar el programa, entre lo que define: Ficheros a compilar, librerías y ejecutables a generar, librerías de las que depende, parámetros adicionales, programas externos a utilizar. Este archivo tiene la capacidad de dividirse, por lo tanto por cada subdirectorio generado se debe crear un CMakeLists, a cada una de estas el CMakeLists principal les debe hacer referencia utilizando el comando ADD_SUBDIRECTORY().

- README, en este archivo se debe describir la forma de compilación del algoritmo, es de ayuda.

3. Una vez que se tiene clara la función de cada archivo, se procede a generar el primer proyecto, para ello es necesario configurar el archivo CMakeLists.txt principal con las siguientes lineas:

   - *set(PARADISEO_ROOT C:/Users/ParadisEO)*, define la ubicación de ParadisEO.
   - *PROJECT(Aquí el nombre)*, define el nombre del proyecto.
   - *SET(PACKAGE_NAME "Nombre" CACHE STRING "Package name" FORCE)*, especifica el nombre del paquete.
   - *SET(PACKAGE_VERSION "Versión" CACHE STRING "Package version" FORCE)*, define la version del paquete.
   - *ADD_SUBDIRECTORY("Directorio")*, agrega los subdirectorios.

4. Luego se configuran los respectivos CMakeLists de los subdirectorios.

5. Se debe agregar el código principal a la carpeta application, para luego compilarlo.

Anexo B

Estructura de directorios

El trabajo de título esta formado por una carpeta principal llamada MyThesis dentro de ésta se encuentran seis subdirectorios. En el directorio "src" se encuentran todos los códigos diseñados como: la función de fitness, el generador de escenarios, los operadores genéticos propuestos, etc. El directorio "datos" está formado por los archivos de configuración .param y los escenarios de pruebas. En el directorio "application" se encuentran los códigos de los algoritmos genéticos propuestos en total son tres archivos .cpp. El directorio "build" es donde se generan los ejecutables. Por último los directorios "doc" y "cmake" vienen por defecto.

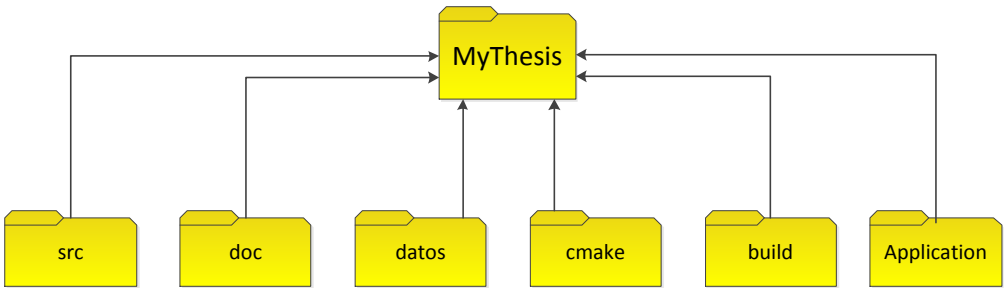


Figura B.1 Estructura de directorio del programa.

Anexo C

Archivo de Configuración

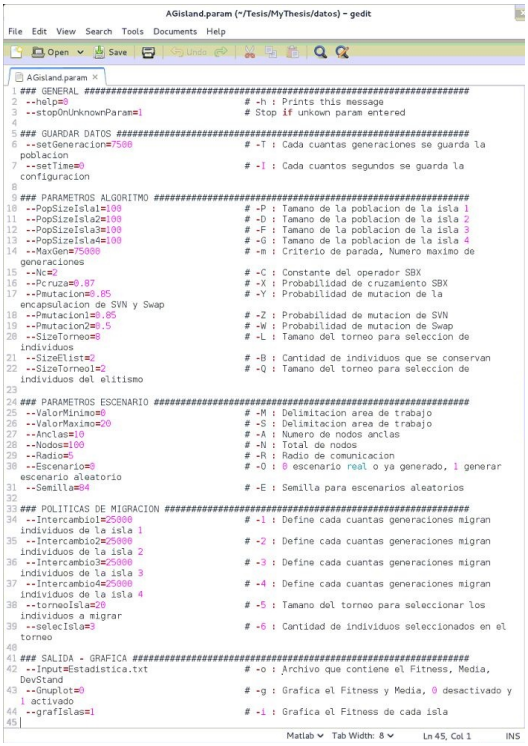


Figura C.1 Archivo de configuración para el modelo de islas.

En la Figura C.1 se muestra el archivo de configuración para el algoritmo genético modelo de islas. Los archivos de configuración de los de más algoritmos son muy similares a éste.

Anexo D

Archivo Datos de Entrada

En las Figura D.1 y Figura D.2 se muestra la estructura de los archivos *"Anclas.txt"* y *"Matriz.txt"*. El primero de ellos está formado con la posición de los nodos anclas. Su estructura está constituida por dos columnas (posiciones x e y) y k filas, donde k es la cantidad de nodos anclas. El segundo archivo está formado por la distancia entre nodos, su estructura es una matriz cuadrada de tamaño n, donde n es el total de nodos.

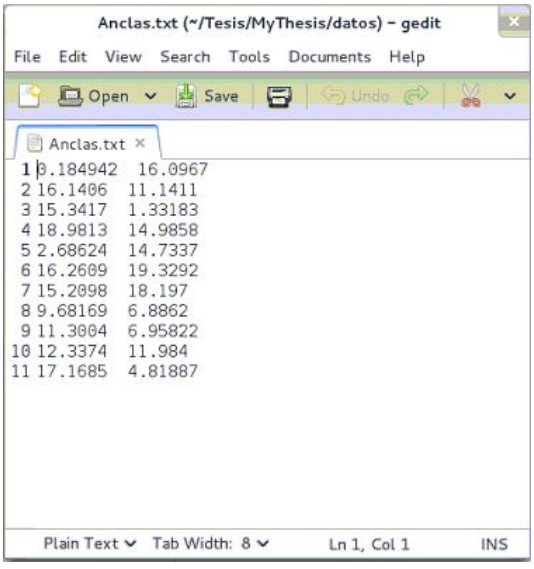


Figura D.1 Estructura del archivo "Anclas.txt".

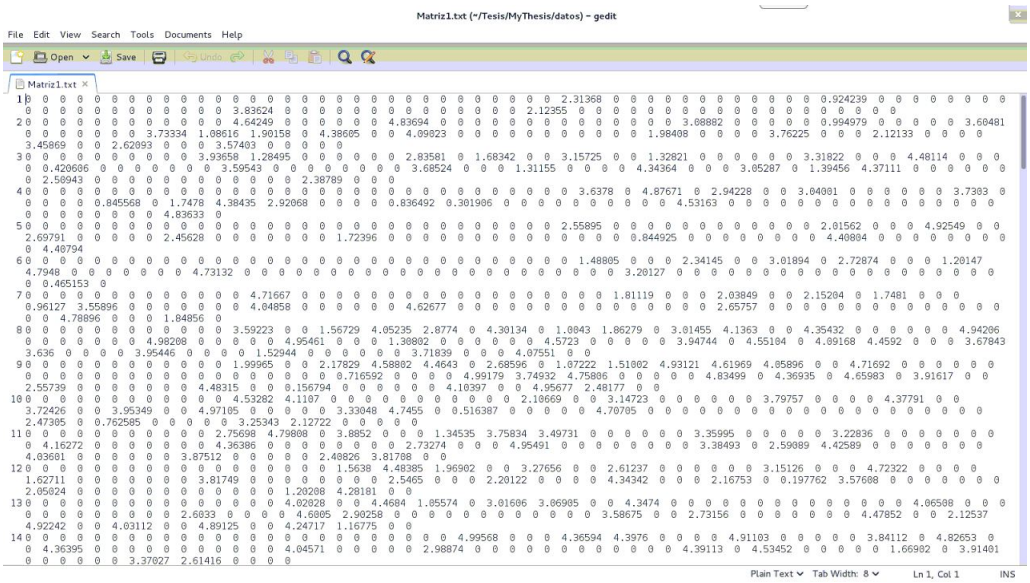


Figura D.2 Estructura del archivo “Matriz.txt”.

Glosario

A

ADN Siglas del ácido desoxirribonucleico. Sustancia química orgánica que porta información genética y se encuentra en el núcleo de toda célula, y que se hereda de padres a hijos., pág. 21.

Alelos Cada uno de los genes del par que ocupa el mismo lugar en los cromosomas homólogos., pág. 22.

C

Código genético es el conjunto de reglas usadas para traducir la secuencia de nucleótidos del ARNm a una secuencia de proteína en el proceso de traducción., pág. 22.

Cromosomas Estructura formada por ADN y proteínas que contiene la información hereditaria., pág. 21.

F

Fenotipo Manifestación visible del genotipo (Genótipo) en un determinado ambiente., pág. 20.

G

Gen Fragmento de ADN que constituye la más pequeña unidad funcional., pág. 22.

Genoma La totalidad de la información genética de un organismo en particular., pág. 21.

Genótipo Conjunto de genes que determinan el fenotipo de los individuos de una especie., pág. 21.

GPS Sistema que permite conocer la posición de un objeto móvil gracias a la recepción de señales emitidas por una red de satélites., pág. 12.

H

Heurística Palabra derivada del griego heuriskein, que significa "encontrar" o "descubrir". Son técnicas que buscan soluciones cercanas al óptimo a un costo computacional razonable, sin garantizar la optimalidad de las mismas, éste tipo de técnicas depende del problema, pág. 11.

L

Locus Punto o lugar del cromosoma en que esta situado un gen., pág. 22.

M

Metaheurística Palabra derivada del griego meta, que significa "más allá" o "nivel superior" y heurítico. Son técnicas que se aplican generalmente a problemas que no tienen un algoritmo o Heurística específica que dé una solución satisfactoria, éste tipo de técnica es independiente del problema., pág. 11.

N

NP Conjunto de problemas que pueden solucionarse en un tiempo polinómico con una máquina de Turing no determinista., pág. 144.

NP-Complete Un problema X es NP-completo si X es NP y NP-hard., pág. 11.

NP-hard Es cualquier tipo de problema, que no necesariamente pertenece al conjunto de la clase NP.

W

WSNs Una red de sensores inalámbrica, es una red ad-hoc que se encuentra formada por pequeños dispositivos que colaboran con uno o más dispositivos conocidos como nodos sensores, y que monitorean un determinado fenómeno en un lugar específico., pág. 6.