INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN Y DESARROLLO DE TECNOLOGÍA DIGITAL

MAESTRÍA EN CIENCIAS EN SISTEMAS DIGITALES

CÓMPUTO DE ALTO RENDIMIENTO PARA RESOLVER GRANDES INSTANCIAS DEL PROBLEMA DEL AGENTE VIAJERO CON UN MODELO DE ISLAS DE ALGORITMOS GENÉTICOS HÍBRIDOS

QUE PARA OBTENER EL GRADO DE

MAESTRÍA EN CIENCIAS EN SISTEMAS DIGITALES

PRESENTA

JÉSUS CARLOS CÁRDENAS PIÑUELAS

BAJO LA DIRECCIÓN DE

DR. JUAN JOSÉ TAPIA ARMENTA

JULIO 2020 TIJUANA, B.C., MÉXICO.

SIP-13

INSTITUTO POLITÉCNICO NACIONAL
SECRETARIA DE IN VESTIGACIÓN Y POSGRADO

ACTA DE REGISTRO DE TEMA DE TESIS
Y DESIGNACIÓN DE DIRECTORES DE TESIS

Tijuana, B.C. a 17 de diciembre del 2018

El Colegio de Profesores de Estudios de Posgrado e Investigación de CITEDI en su sesión ordinaria No. 12/18 celebrada el día 13 del mes de diciembre, 2018 conoció la solicitud presentada por el(la) alumno(a):

CARDENAS	PIÑUELAS	JESUS CARLOS
Apellido paterno	Apellido materno	Nombre (s)

Con registro: B 1 8 0 3 7 4

Aspirante de: **MAESTRÍA EN CIENCIAS EN SISTEMAS DIGITALES**

1.- Se designa al aspirante el tema de tesis titulado:
Cómputo de alto rendimiento para resolver grandes instancias del problema del agente viajero con un modelo de islas de algoritmos genéticos híbridos.

De manera general el tema abarcará los siguientes aspectos:
Proponer e implementar algoritmos con cómputo de alto rendimiento para resolver grandes instancias del problema del agente viajero con un modelo de islas de algoritmos genéticos híbridos.

2.- Se designa como Directores de Tesis a los Profesores:
DR. JUAN JOSÉ TAPIA ARMENTA

3.- El trabajo de investigación base para el desarrollo de la tesis será elaborado por el alumno en:
CENTRO DE INVESTIGACIÓN DE DESARROLLO DE TECNOLOGÍA DIGITAL
que cuenta con los recursos e infraestructura necesarios.

4.- El interesado deberá asistir a los seminarios desarrollados en el área de adscripción del trabajo desde la fecha en que se suscribe la presente hasta la aceptación de la tesis por la Comisión Revisora correspondiente:

Director de Tesis

DR. JUAN JOSÉ TAPIA ARMENTA

Aspirante

ING. JESUS CARLOS CARDENAS PIÑUELAS

Presidente del Colegio

DR. JULIO CÉSAR ROLÓN GARRIDO

S.E.P.
INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN Y DESARROLLO DE TECNOLOGÍA DIGITAL
DIRECCIÓN

SIP-14
REP 2017

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de Tijuana B.C. siendo las 14:00 horas del día 20 del mes de Julio del 2020 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Posgrado del: Centro de Investigación y Desarrollo de Tecnología Digital para examinar la tesis titulada: Cómputo de alto rendimiento para resolver grandes instancias del problema del agente viajero con un modelo de islas de algoritmos genéticos híbridos.

del (la) alumno (a):

Apellido Paterno:	Cardenas	Apellido Materno:	Piñuelas	Nombre (s):	Jesus Carlos

Número de registro: B 1 8 0 3 7 4

Aspirante del Programa Académico de Posgrado: Maestría en Ciencias en Sistemas Digitales

Una vez que se realizó un análisis de similitud de texto, utilizando el software antiplagio, se encontró que el trabajo de tesis tiene _7_ % de similitud. **Se adjunta reporte de software utilizado.**

Después que esta Comisión revisó exhaustivamente el contenido, estructura, intención y ubicación de los textos de la tesis identificados como coincidentes con otros documentos, concluyó que en el presente trabajo **SI** [ ] **NO** [x] **SE CONSTITUYE UN POSIBLE PLAGIO.**

JUSTIFICACIÓN DE LA CONCLUSIÓN: *(Por ejemplo, el % de similitud se localiza en metodologías adecuadamente referidas a fuente original)*

El porcentaje de similitudes generado por Turnitin es pequeño y corresponde al texto genérico de una tesis de posgrado y del tema de tesis del estudiante. Los resultados fueron obtenidos mediante software realizado por el estudiante bajo la supervisión de su director de tesis.

****Es responsabilidad del alumno como autor de la tesis la verificación antiplagio, y del Director o Directores de tesis el análisis del % de similitud para establecer el riesgo o la existencia de un posible plagio.**

Finalmente, y posterior a la lectura, revisión individual, así como el análisis e intercambio de opiniones, los miembros de la Comisión manifestaron **APROBAR** [x] **SUSPENDER** [ ] **NO APROBAR** [ ] la tesis por **UNANIMIDAD** [x] o **MAYORÍA** [ ] en virtud de los motivos siguientes:

Es un trabajo de investigación original y el algoritmo desarrollado contribuye al conocimiento en el tema de tesis del estudiante

COMISIÓN REVISORA DE TESIS

Dr. Juan José Tapia Armenta
Director de Tesis
Nombre completo y firma

Dr. Oscar Humberto Montiel Ross
Nombre completo y firma

Dr. Moisés Sánchez Adame
Nombre completo y firma

Dr. Ciro Andrés Martínez García Moreno
Nombre completo y firma

Dr. Leonardo Trujillo Reyes
Nombre completo y firma

Dr. Julio César Rolón Garrido
Nombre completo y firma
PRESIDENTE DEL COLEGIO DE PROFESORES

INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN Y DESARROLLO DE TECNOLOGÍA DIGITAL
DIRECCIÓN

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de Tijuana, Baja California el día 18 del mes Julio del año 2020, el (la) que suscribe Ing. Jesús Carlos Cárdenas Piñuelas alumno (a) del Programa de Maestría en Ciencias en Sistemas Digitales con número de registro B180374, adscrito a Centro de Investigación y Desarrollo de Tecnología Digital, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de Dr. Juan José Tapia Armenta y cede los derechos del trabajo titulado CÓMPUTO DE ALTO RENDIMIENTO PARA RESOLVER GRANDES INSTANCIAS DEL PROBLEMA DEL AGENTE VIAJERO CON UN MODELO DE ISLAS DE ALGORITMOS GENÉTICOS HÍBRIDOS, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

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

Ing. Jesús Carlos Cárdenas Piñuelas

Nombre y firma

Agradecimientos

Al Consejo Nacional Mexicano de Ciencia y Tecnología (CONACYT), por su invaluable apoyo y patrocinio para la realización de este trabajo de tesis.

Al Instituto Politécnico Nacional (IPN) y al Centro de Investigación y Desarrollo de Tecnología Digital, por recibirme como parte de su estimada comunidad.

Al Dr. Juan José Tapia Armenta, por su apoyo y guía a lo largo de la realización del presente trabajo de investigación.

Al comité evaluador de tesis, por sus acertados comentarios y correcciones que a lo largo de mi proceso de investigación me ayudaron a mejorar mis resultados y presentaciones.

A mi familia, por ser un pilar fundamental en mi vida y una gran motivación para continuar creciendo personal y profesionalmente.

A la comunidad de CITEDI en general, por su valioso apoyo brindado durante el tiempo transcurrido en dicho centro.

CÓMPUTO DE ALTO RENDIMIENTO PARA RESOLVER GRANDES INSTANCIAS DEL PROBLEMA DEL AGENTE VIAJERO CON UN MODELO DE ISLAS DE ALGORITMOS GENÉTICOS HÍBRIDOS

Resumen

En esta investigación se presenta el desarrollo de un algoritmo genético paralelo para resolver el problema del agente viajero. Las principales características del algoritmo son la implementación del operador de cruce con ensamble de aristas (EAX) combinado con un método de búsqueda local (2-opt) y la aplicación del algoritmo genético paralelo con un modelo de islas. Además, la población inicial se genera con un algoritmo de búsqueda local muy eficiente.

El algoritmo obtenido se paralelizó en un procesador multinúcleo con memoria compartida utilizando el estándar de hilos POSIX en lenguaje C. En la paralelización desarrollada se implementó un hilo de ejecución para cada isla. Los principales pasos del algoritmo se desarrollaron en paralelo, y después de cada paso se hizo la sincronización correspondiente. En cada generación se calcula una cantidad de descendientes previamente definida para cada uno de los individuos, después se selecciona al descendiente de mejor calidad para reemplazar al individuo padre en caso de ser mejor. Para la migración de individuos se escoge el mejor individuo de una isla que sustituye al peor individuo de otra isla, mediante una topología de anillo.

El algoritmo se ejecutó para instancias del problema del agente viajero en la escala de decenas de miles de ciudades, obtenidas de la biblioteca TSPLIB, las mejores soluciones obtenidas están muy cercanas a los valores óptimos reportados en dicha biblioteca. Los resultados indican que el utilizar un modelo de isla acelera la convergencia.

Palabras clave: Problema del agente viajero, Algoritmo genético paralelo, Operador EAX.

HIGH PERFORMANCE COMPUTING TO SOLVE LARGE INSTANCES OF THE TRAVELING SALESMAN PROBLEM WITH A ISLAND MODEL OF HYBRID GENETIC ALGORITHM

Abstract

This research presents the development of a parallel genetic algorithm to solve the traveling salesman problem. The main features of the algorithm are the implementation of the edge assembling crossover operator (EAX) combined with a local search method (2-opt) and the application of the parallel genetic algorithm with an island model. Furthermore, the initial population is generated with a very efficient local search algorithm.

The algorithm obtained was parallelized in a multicore processor with shared memory using the POSIX threads standard in C language. In the developed parallelization, an execution thread was implemented for each island. The main steps of the algorithm were developed in parallel, and after each step the corresponding synchronization was made. In each generation, a predefined quantity of descendants is calculated for each of the individuals, then the best quality descendant is selected to replace the parent individual if better. For the migration of individuals, the best individual from one island is chosen to replace the worst individual from another island, using a ring topology.

The algorithm was executed for instances of the traveling salesman problem on the scale of tens of thousands of cities, obtained from the TSPLIB library, the best solutions obtained are very close to the optimal values reported in said library. The results indicate that using an island model accelerates convergence.

Keywodrs: Traveling salesman problem, Parallel genetic algorithm.

Índice general

Lista de Figuras

Lista de Tablas

Lista de Algoritmos

Capítulo 1

Introducción

La solución de problemas de optimización combinatoria tiene muchas aplicaciones en diferentes áreas del conocimiento, ya sea minimizar el consumo de ciertos recursos u obtener el mayor beneficio posible, siempre se traduce esto en optimizar una cierta relación de variables a la que se le llama función objetivo, cumpliendo ciertas reglas denominadas restricciones.

Particularmente, el problema del agente viajero TSP (del inglés Travelling Salesman Problem) es un problema de optimización combinatoria que aborda el problema de encontrar la ruta más corta en un conjunto de ciudades, de manera que se visite una sola vez cada ciudad y se regrese a la ciudad de inicio al final del recorrido. La solución al TSP implica un alto costo computacional tanto en tiempo como en recursos. Para un método de búsqueda exhaustiva podría ser una tarea imposible de cumplir, si se considera un límite de tiempo aceptable.

Por otro lado, las metaheurísticas ofrecen aproximaciones de buena calidad en tiempos razonables de ejecución, mejorando considerablemente su rendimiento si se aprovechan los avances tecnológicos respecto a procesadores. En la actualidad se cuenta con arquitecturas de multiprocesadores, procesadores multinúcleo y unidades de procesamiento gráfico o GPU (del inglés Graphics Processing Unit) que permiten procesar grandes cantidades de información en tiempos de ejecución razonables.

Esta nueva generación de computadoras eficientes ha conducido al desarrollo de un nuevo modelo de programación que se denomina programación en paralelo, considerado también como cómputo de alto rendimiento; haciendo referencia al procesamiento de datos en diferentes unidades al mismo tiempo. Este modelo resulta ideal para solucionar problemas que requieran una gran cantidad de reciursos de procesamiento, entre los cuales no existe dependencia de datos.

Para efectos de esta investigación se abordará el desarrollo de algoritmos genéticos (AG), los cuales pueden ser clasificados como métodos metaheurísticos dentro del área de optimización. Los algoritmos genéticos están inspirados en la evolución genética, y su función en esencia es hacer evolucionar cierta población de individuos bajo las operaciones de cruza y mutación.

Los AG pueden ser aplicados en problemas de optimización que se modelan como una función objetivo discontinua o dinámica, problemas para los cuales un método determinístico es poco eficiente.

Un AG puede ser implementado fácilmente de forma paralela debido a que procesa cada individuo de la población de manera independiente. Un algoritmo genético paralelo (AGP) se puede programar para trabajar en las arquitecturas de multiprocesadores. Al combinarse un AG con un método de búsqueda local se obtiene un algoritmo genético híbrido que puede mejorar la eficiencia de un AG clásico; al paralelizar este tipo de algoritmos obtenemos un algoritmo genético paralelo híbrido (AGPH).

Se plantea desarrollar un algoritmo genético paralelo híbrido con modelo de islas (AGPH-I) representativo de la problemática anteriormente señalada para su posterior análisis. En el desarrollo de este algoritmo se aplicará una implementación del operador de cruce con ensamble de aristas (EAX), mismo que genera soluciones de muy buena calidad en tiempos de ejecución relativamente cortos.

Una vez desarrollado el algoritmo AGPH-I he implementado en computadoras multinúcleo, se comparado con un algoritmo genético paralelo híbrido orientado a objetos (AGPH-O) respecto a la generación de la población inicial y el tiempo de ejecución. De la misma manera se hace una comparación del algoritmo AGPH-I con un algoritmo genético híbrido secuencial (AGH-S) y un algoritmo genético paralelo híbrido con la población global contenida en una isla (AGPH-P), con el fin de medir la eficiencia del algoritmo AGPH-I respecto a diferentes implementaciones. Se realizaron experimentos con diferentes instancias del TSP para después analizar los resultados obtenidos con las implementaciones desarrolladas.

1.1. Trabajos relacionados con el estado del arte

La solución del TSP tienen aplicación en una gran cantidad de áreas de estudio de la industria y en la sociedad en general, por lo cual existen incontables trabajos de investigación orientados a esta temática. En esta sección se mencionan los resultados presentados en algunos de estos trabajos.

Un estudio de la paralelización de los AG utilizando el modelo de islas en un entorno informático heterogéneo asíncrono se presenta en [1], basado en un análisis estadístico de un conjunto completo de experimentos, utilizando instancias de la biblioteca TSPLIB del mundo real, el estudio investiga la pregunta: ¿Cuál es la topología de isla más rápida para el algoritmo genético basado en órdenes, en un entorno de computación basado en una malla heterogénea distribuida asíncrona? Además, se propone un nuevo índice de aceleración. Se considera una vasta diversidad de tipos y características de topologías. Se considera también un número diferente de

nodos. La topología de anillo de árbol coordinada es una novedad. Este tipo de topologías nos permiten evaluar tres casos notables, sin migración, migración hacia una isla elite y migración hacia y desde la isla elite.

Un algoritmo mejorado para aproximar el TSP en gráficos simétricos totalmente conectados mediante la utilización de la GPU se presenta en [2]. Este enfoque mejora un algoritmo existente 2-opt hill-climbing con reinicios aleatorios al considerar múltiples actualizaciones a la ruta actual encontrada en paralelo, y permite un número k de actualizaciones por iteración, llamado k-swap. Con esta modificación k-swap, se muestra una aceleración sobre el algoritmo existente de 4.5x a 22.9x en conjuntos de datos que van desde 1,400 a 33,810 nodos, respectivamente.

En el trabajo descrito en [3] se presenta un estudio experimental que ha llevado a cuatro conclusiones principales: el algoritmo puro de Lin-Kernighan funciona bien en casos de pequeña y mediana escala; el algoritmo puro de Lin-Kernighan es mejor que la búsqueda en vecindarios y los métodos de computación evolutiva; los algoritmos híbridos propuestos tienen un rendimiento mucho mejor que los métodos de búsqueda local pura y computación evolutiva; la mejor configuración de la optimización de colonias de hormigas basada en la población híbrida siempre es mejor que cualquier configuración de algoritmos evolutivos híbridos. Los experimentos informados en este documento se ejecutaron en el sistema de súper cómputo en el Centro de Supercomputación de la Universidad de Ciencia y Tecnología de China.

El trabajo presentado en [4] concluye que la configuración de parámetros predeterminada de Concorde, LKH (del inglés Lin-Kernighan Heuristic) y EAX puede beneficiar el desempeño de cada uno de ellos. Señalando que recientemente se ha demostrado que este es el caso de EAX. La configuración automática no ha producido mejoras similares para LKH y Concorde, por lo que puede valer la pena examinar más cuidadosamente cómo la configuración automática puede afectar el comportamiento de los mismos.

Un AG para encontrar soluciones de muy alta calidad para el TSP se propone en [5], el cual ha encontrado soluciones óptimas para la mayoría de las instancias de referencia. Una de las fortalezas de este AG es el uso de EAX, un operador de alto potencial para el TSP. La utilización del EAX reduce significativamente el costo computacional. Se ha propuesto un algoritmo AG que mantiene efectivamente la diversidad de la población.

En el artículo [6] se propone un algoritmo genético paralelo con una implementación simple del EAX, el cual logra una velocidad mayor en comparación al AG con EAX presentado en [5] sin disminuir la calidad de las soluciones, a través de experimentos numéricos en instancias a escala de 100,000 ciudades. Se encontraron también nuevos recorridos en instancias de 120,000 ciudades y 180,000 ciudades.

En el trabajo del 2018 presentado en [7] se experimenta con instancias del TSP a pequeña y gran escala, las instancias utilizadas se encuentran en el rango de 29 ciudades (instancia

bayg29) y 7397 ciudades (instancia Pla7397). Este rango de instancias queda por debajo del límite superior del rango presentado en este trabajo de tesis, el cual tiene como instancia de mayor tamaño a RBZ43748.

1.2. Planteamiento del problema

El TSP es un problema NP-Difícil; como un problema de decisión pertenece a la clase NP-Completo. En el peor caso el tiempo de ejecución para resolver el TSP aumenta de forma factorial $O(N!)$ con respecto a la cantidad N de ciudades. Se sabe que fácilmente se puede comprobar si una solución es correcta, pero es difícil encontrar una buena solución.

El problema de investigación consiste en resolver instancias del TSP con un tamaño del orden de decenas de miles de ciudades, aplicando un algoritmo genético híbrido con un modelo de islas y el operador de cruce EAX.

1.3. Hipótesis nula

No es posible resolver instancias en la escala de decenas de miles de ciudades del problema TSP, con un error menor al 2 %.

1.4. Hipótesis

Con un algoritmo genético paralelo híbrido con modelo de islas y la implementación del operador genético EAX es posible resolver instancias en la escala de decenas de miles de ciudades del problema TSP, con un error menor al 2 %.

1.5. Objetivo General

Proponer e implementar un algoritmo con cómputo de alto rendimiento para resolver grandes instancias del problema del agente viajero con un modelo de islas de algoritmos genéticos híbridos.

1.5.1. Objetivos específicos

- Conocer los avances más recientes para la solución del problema del agente viajero con algoritmos genéticos.

- Comprender el operador EAX y el cómputo paralelo para aplicarlos en la solución del problema del agente viajero.
- Proponer un algoritmo genético paralelo híbrido con el operador EAX para resolver el problema del agente viajero.
- Implementar el algoritmo con el operador EAX para resolver instancias del problema del agente viajero con más de 10000 ciudades.
- Analizar los resultados obtenidos para comprobar la hipótesis y establecer las conclusiones.

1.6. Organización de la tesis

El resto del documento está organizado de la siguiente manera. En el Capítulo 2 se describe el problema del agente viajero, se explica el concepto de complejidad computacional y se describen algunos métodos que resuelven el TSP. En el Capítulo 3 se describe el marco teórico de los algoritmos genéticos detallando los algoritmos genéticos para permutaciones, mismos que corresponden al TSP. La metodología para la implementación de un algoritmo genético paralelo con el operador EAX para resolver el TSP se presenta en el Capítulo 4. En el Capítulo 5 se muestran los resultados de la solución de diversas instancias del TSP. Finalmente, en el Capítulo 6 se presentan los resultados y recomendaciones para trabajo futuro.

Capítulo 2

El Problema del agente viajero

El TSP fue formulado por los matemáticos W.R. Hamilton y Thomas Kirkman en 1800 y se centra en encontrar la ruta óptima (de costo mínimo) para visitar un número determinado de ciudades [8]; es decir, dado un conjunto de ciudades y las distancias entre cada par de ellas, encontrar la ruta más corta que visite cada ciudad una vez y al final regrese a la ciudad inicial.

El TSP se puede formular de la siguiente manera: Sea C la matriz de distancias donde el elemento C_{ij} de C corresponde a la distancia entre la ciudad i y la ciudad j. Sea P_N la colección de todas las permutaciones π del conjunto $\{1, 2, ..., N\}$, entonces la solución del TSP consiste en encontrar en P_N una permutación $\pi = ((\pi_1), (\pi_2), ..., (\pi_N))$ de tal manera que el costo

$$d = \sum_{i=1}^{N-1} C_{\pi(i)\pi(i+1)} + C_{\pi(N)\pi(1)} \tag{2.1}$$

sea mínimo. El TSP se divide en dos clases de acuerdo a la naturaleza de la matriz de distancias: simétrico y asimétrico. En el TSP simétrico la distancia C_{ij} es la misma que la distancia C_{ji}, en caso contrario es asimétrico. Lo cual significa que, en la representación con grafos, el TSP simétrico es un grafo no dirigido y el TSP asimétrico se representa con un grafo dirigido.

El campo de aplicación del TSP es muy extenso y variado, algunas aplicaciones son:

- Diseño de rutas. Optimización de rutas para la distribución de productos o servicios a varios clientes.
- Circuitos integrados. Minimizar el tiempo requerido para grabar una secuencia de líneas que forman conexiones eléctricas entre los componentes de un circuito.
- Circuitos perforados. Minimizar la distancia que debe recorrer una máquina perforadora para completar la tarea de perforar un circuito previamente diseñado.

2.1. Formulación del problema del agente viajero

Sea $G = (V, E)$ un grafo, F la familia de todos los ciclos (recorridos) en G y para cada arista $e \in E$ el costo C_e; el TSP consiste en encontrar un recorrido (ciclo Hamiltoniano) en G de tal manera que la suma de los costos de las aristas del recorrido sea lo más pequeña posible. Suponer que G es un grafo completo y al establecer el nodo $V = 1, 2, ..., n$; C es la matriz de costos $C = (c_{ij})_{nxn}$, donde c_{ij} corresponde al costo de la arista que une al nodo i con el nodo j en G [9].

Un algoritmo que asegura encontrar el óptimo es el de búsqueda exhaustiva, es decir encontrar el óptimo al probar todas las posibilidades, este algoritmo no es práctico ya que su complejidad computacional es $N!$.

Para visitar la primer ciudad se tienen N posibilidades, para la segunda ciudad $(N-1)$, para la tercer ciudad $(N-2)$ y así sucesivamente hasta llegar a la última ciudad, que solo se tiene una posibilidad. De esta manera el total de posibilidades para la ruta óptima es $N!$. Además, como el problema es simétrico, significa que hay N rutas repetidas, y al considerar que la ruta forma un ciclo al conectar la última ciudad visitada con la primera, entonces para cada ruta hay otra con la misma distancia al recorrerse en el sentido inverso. Por lo tanto la cantidad de posibilidades para el TSP es $\frac{(N-1)!}{2}$, lo que significa que la complejidad computacional es de $O(N!)$.

Esto implica que resolver el TSP por un algoritmo de búsqueda exhaustiva no es viable. El estudio de formas alternativas de resolución se basa en gran medida en el diseño de metaheurísticas que proporcionen un cierto equilibrio entre el tiempo de ejecución y la calidad de la solución obtenida. Para efectos del presente trabajo se considerará un modelo simétrico del mencionado problema, es decir, la distancia entre cada par de ciudades es la misma en ambas direcciones, formando un grafo no dirigido, ésto reduce el número de soluciones posibles a $\frac{(N-1)!}{2}$.

2.2. Complejidad Computacional

La complejidad computacional estudia el crecimiento del costo computacional al resolver un problema en relación al crecimiento de dicho problema [10]. Esta rama de la teoría computacional cuenta con diferentes métricas para evaluar el aumento del costo computacional con respecto al tamaño de un problema; una de ellas es llamada tiempo polinomial, de la cual podemos distinguir logarítmicos $O(\log N)$, lineales $O(N)$, cuadráticos $O(N^2)$, cúbicos $O(N^3)$, etc. Por ejemplo, si a una computadora necesita 1 segundo para ordenar mil números, 10 segundos para ordenar 10 mil números, 20 para 20 mil, etc.; la complejidad del problema es lineal, la función del tiempo con respecto al tamaño del problema es lineal. Por otro lado, en un algoritmo que

implementa ciclos anidados, el nivel de complejidad (tiempo polinómico) aumenta conforme aumentan la cantidad de ciclos involucrados; para dos ciclos anidados la complejidad es $O(N^2)$, para tres $O(N^3)$, y así sucesivamente.

Una vez identificada la complejidad computacional de un algoritmo, este puede ser clasificado de acuerdo a los conjuntos P, NP, NP-Completo y NP-Difícil. La clase P son aquellos problemas que pueden resolverse en el tiempo $O(N^k)$ para alguna constante k, donde N es el tamaño de la entrada al problema; La clase NP es el conjunto de problemas de los cuales se puede comprobar en un tiempo razonable si una respuesta es correcta; cualquier problema en P también está en NP, ya que si un problema está en P entonces podemos resolverlo en tiempo polinómico. Un problema está en la clase NP-Completo si está en NP y es tan difícil como cualquier problema en NP; si cualquier problema NP-Completo puede resolverse en tiempo polinómico entonces cada problema NP-Completo tiene un algoritmo de tiempo polinómico [11].

La clase NP-Difícil contiene a los problemas de decisión tan difíciles como un problema de NP; si se puede encontrar un algoritmo que resuelva un problema en NP-Difícil en tiempo polinómico es posible construir un algoritmo de tiempo polinómico para cualquier problema de NP.

Una representación gráfica de la clasificación de problemas con respecto a la teoría de la complejidad computacional se presenta en la Fig. 2.1 ([10]).

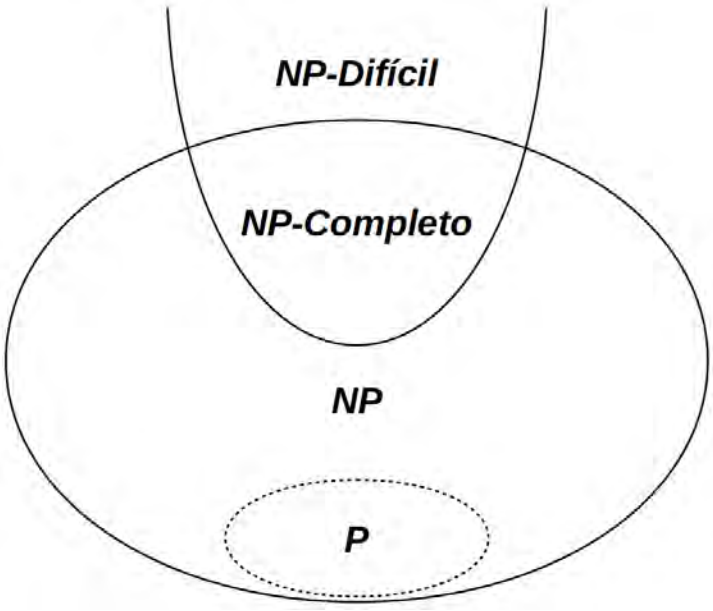


Figura 2.1: Representación gráfica de los conjuntos de complejidad computacional.

El TSP es un problema NP-Difícil en optimización combinatoria; la versión de decisión del mismo (dado un costo C, decidir que grafo tiene menor costo que C) pertenece a la clase

NP-Completo. En el peor caso el tiempo de ejecución para resolver el TSP aumenta de forma factorial $O(N!)$ con respecto al número de ciudades. Por otro lado, se sabe que el problema del TSP se encuentra en el conjunto de problemas NP más no en el conjunto P; es decir, fácilmente se puede comprobar si una buena solución para el TSP es correcta, pero es difícil (costoso) encontrar una buena solución conforme crece el tamaño del problema.

2.3. Métodos clásicos para resolver el problema del agente viajero

Los algoritmos de fuerza bruta son una primera solución para el TSP, consisten en generar todas las soluciones candidatas para luego evaluar cada una de ellas. El inconveniente es el enorme costo computacional que conlleva, pues la cantidad de posibles soluciones a generar crece de forma factorial ($N!$) al aumentar el tamaño del problema.

Una estrategia intuitiva para el TSP es el algoritmo del vecino más cercano, que sigue una estrategia del algoritmo *greedy* (tipo avaricioso), en el cual el agente viajero debe ir siempre a la ciudad más cercana no visitada. La complejidad de este método es $O(N^2)$ [12] y se usa principalmente para generar la población inicial en un algoritmo evolutivo.

El algoritmo de Lin-Kernighan considera un conjunto creciente de posibles intercambios de nodos. Estos intercambios se eligen de tal manera que se pueda formar un recorrido factible en cualquier etapa del proceso. Si se consigue un nuevo recorrido más corto, el recorrido inicial se reemplaza por el nuevo recorrido [13].

Concorde es un software para resolver el TSP que está escrito en el lenguaje de programación ANSI C. Se ha utilizado para obtener las soluciones óptimas en instancias de la biblioteca TSPLIB de hasta 85.900 ciudades. La biblioteca de Concorde incluye más de 700 funciones que permiten crear códigos especializados para problemas similares al TSP. Todas las funciones de Concorde son aptas para la programación en paralelo con memoria compartida; incluye código para ejecutarse en redes de estaciones de trabajo UNIX. Hans Mittelmann ha creado un servidor NEOS para Concorde, que permite a los usuarios resolver instancias de TSP en línea [14].

Los algoritmos genéticos son otra solución para el TSP, estos trabajan con una población de individuos que va evolucionando a lo largo del tiempo de forma que los individuos mejor adaptados tendrán más probabilidades de sobrevivir. Un factor muy importante en los AG es la capacidad de los individuos para reproducirse. El rendimiento del algoritmo depende del modelado del problema.

Capítulo 3

Algoritmos genéticos paralelos

3.1. Algoritmos genéticos

Los AG son métodos de búsqueda basados en principios de selección natural y recombinación, buscan la solución de un problema en un conjunto de soluciones candidatas. Se evalúan las soluciones y se seleccionan las mejores para generar cierta cantidad de generaciones en un lapso de tiempo. Los buenos rasgos prevalecen durante varias generaciones mejorando la calidad de las soluciones. En los AG la mutación se considera un operador secundario cuya función es garantizar diversidad en la población [15].

En un AG clásico los individuos son cadenas de bits y la población consta de cadenas generadas al azar en la generación inicial, mientras que en problemas de optimización combinatoria son listas de símbolos, la representación de la solución se llama genotipo o cromosoma [16]. La cantidad de individuos influye para encontrar buenas soluciones y el tiempo necesario para lograrlo, y se busca lograr un equilibrio entre la calidad de la solución y el tiempo necesario para encontrarla. Cada individuo tiene un valor de aptitud que representa la calidad de la solución, los AG son métodos de optimización y la calidad de una solución es la función objetivo evaluada con la solución que representa un individuo, dicho valor es utilizado para seleccionar a los individuos que construirán próximas generaciones.

3.1.1. Funcionamiento de un algoritmo genético básico

Los algoritmos genéticos pueden presentar variaciones, dependiendo de la manera en que se aplican sus operadores, de cómo se seleccionan los individuos y de cómo se define la nueva población.

En un AG cada iteración como parte de una generación) consta de los siguientes pasos [17]:

- **Selección.** El primer paso consiste en seleccionar individuos para la reproducción. Esta

selección se realiza al azar con una probabilidad que depende de la aptitud relativa de los individuos, de modo que a menudo se eligen los mejores para reproducción.

- **Reproducción.** En el segundo paso, los descendientes son generados por los individuos seleccionados. Para generar nuevos cromosomas, el algoritmo puede usar tanto recombinación como mutación.
- **Evaluación.** Se evalúa la aptitud de los nuevos cromosomas.
- **Reemplazo.** Durante este paso los individuos de la población anterior son eliminados y reemplazados por nuevos individuos con mejor aptitud mediante una selección elitista (estrategia generacional de reemplazo).

El algoritmo se detiene cuando la población converge hacia la solución óptima.

En el proceso de selección se escogen los individuos que se van a reproducir. Regularmente se otorga mayor oportunidad de reproducción a los individuos más aptos, por lo cual la selección depende de la calidad de los individuos. Sin embargo, no se deben descartar por completo a los individuos menos aptos ya que la población se volvería homogénea de forma prematura. Dos de los métodos de selección más frecuentes son:

- Selección por ruleta. La selección por ruleta es un método muy frecuente en AG y evolutivos. Las implementaciones existentes seleccionan uno de N individuos usando algoritmos de búsqueda de complejidad $O(N)$ ú $O(logN)$ [18]. A cada individuo se le asigna una probabilidad de selección proporcional a su aptitud, de manera que la suma de todos los porcentajes sea 1. Los mejores individuos reciben mayor probabilidad. Se generar un número aleatorio en el intervalo $[0, 1]$ y se selecciona al individuo correspondiente a dicho número.
- Selección por torneo. Se selecciona al azar una cantidad de individuos, de los cuales se selecciona el más apto para pasarlo a la siguiente generación. Al variar el número de individuos de cada torneo se modificar la presión de selección; cuando participan muchos la presión de selección aumenta y los peores individuos apenas tienen oportunidad de reproducirse, cuando el tamaño del torneo es reducido la presión de selección disminuye y los peores individuos tienen más oportunidad de ser seleccionados.

Una vez seleccionado un par de individuos se procede con la operación de cruce. En este proceso se generan puntos de cruce aleatorios a partir de los cuales se intercambian subcadenas de valores para generar nuevos individuos que son candidatos a formar parte de la nueva generación, el operador de cruce clásico genera un punto de cruce para realizar el intercambio de valores. En la Fig. 3.1 se muestra un ejemplo del funcionamiento del operador de cruce.

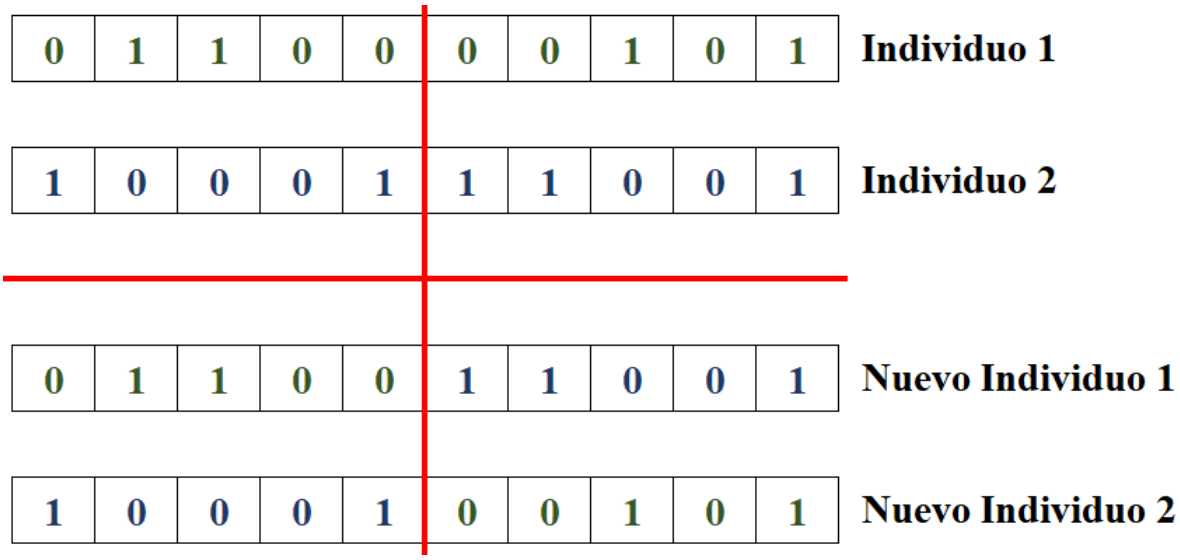


Figura 3.1: Cruce clásico en soluciones con representación binaria.

El operador de mutación genera cambios aleatorios en los nuevos individuos para mantener diversidad en la población. Para cada gen del cromosoma, el AG calcula una probabilidad de cambio que, al estar dentro del rango de la probabilidad de mutación previamente definida, cambia el valor del gen correspondiente. En la Fig. 3.2 se muestra un ejemplo del funcionamiento de este operador.

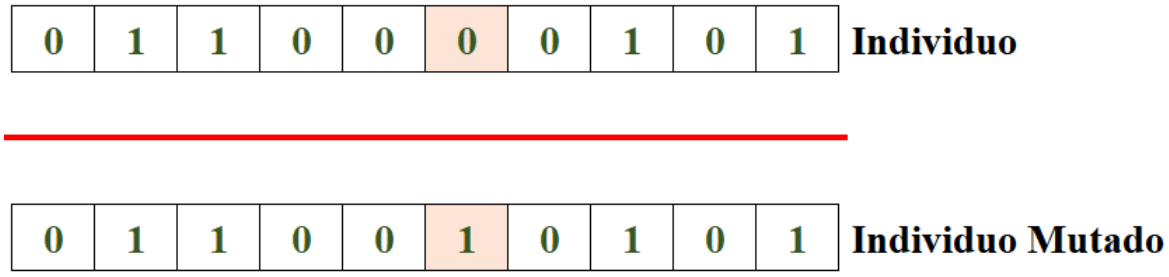


Figura 3.2: Mutación clásica en soluciones con representación binaria.

3.1.2. Ventajas y limitaciones de un algoritmo genético

De entre las ventajas de los algoritmos genéticos destaca su naturaleza paralela, lo que significa que pueden procesar muchas soluciones posibles de forma simultánea en vez de seguir un proceso secuencial como se emplea en otros procesos de solución. Técnicas tradicionales exploran el conjunto solución en una sola dirección con una gran probabilidad de caer en un óptimo local. Cuando se utilizan algoritmos genéticos para solucionar problemas de optimización

es mucho menos probable caer en óptimos locales.

Cuando se resuelve un problema con un AG no se requiere tener conocimientos muy específicos del problema ya que realizan cambios en su espectro de soluciones y después deciden si se ha producido una mejora, a través de su función de aptitud; además, su implementación en las arquitecturas computacionales en paralelo resulta sencilla.

Por otro lado, se debe tener una clara definición del problema; dependiendo de los parámetros establecidos puede tardar mucho tiempo en converger a una solución o simplemente no converger. También se puede presentar el caso de que converja de forma prematura cuando un individuo es dominante sobre el resto de la población.

3.2. Algoritmos genéticos para permutaciones

Para resolver el TSP con un AG se debe considerar que la cadena de valores que representa a una solución debe contener valores no repetidos, ya que cada valor en la cadena representa un nodo o ciudad. El repetir un valor significa que la ruta generada visita dos veces o más una ciudad, mientras deja de visitar a otras. En esta investigación las soluciones se representan con sucesiones de números enteros y un ejemplo de esto es el siguiente; si el agente viajero debe visitar cinco ciudades una ruta es $1, 2, 3, 4, 5$ donde cada número representa una ciudad; sin embargo, la secuencia $1, 2, 3, 2, 5$ no es una ruta valida ya que la ruta generada visitará dos veces la ciudad 2 y ninguna vez la ciudad 4.

Para abordar este problema se representan las soluciones como permutaciones. Se define como permutaciones P_n^m a las distintas maneras en que se pueden ordenar m elementos en n posiciones siempre y cuando $m \geq n$; en este ordenamiento importa el orden, ya que el intercambio entre dos elementos distintos genera una nueva permutación; y no se puede repetir elementos [19]. Se puede calcular la cantidad de permutaciones en base a m y n como

$$P_n^m = \frac{m!}{(m-n)!}. \tag{3.1}$$

En el problema del TSP se deben ordenar n nodos en n posiciones resultando en una cantidad de permutaciones igual a $P_n^n = P_n = n!$. Esta representación de las soluciones implica que el operador de cruce de un AG para el TSP no puede ser un operador para representaciones clásicas, como las representaciones binarias; es decir, es necesario diseñar operadores de cruce especiales para soluciones que son permutaciones.

3.2.1. Cruce para permutaciones

La representación de permutaciones se usa frecuentemente en problemas de optimización combinatoria como el TSP, y consiste en usar cadenas de enteros para representar una permutación. Al efectuar cualquiera cruce entre dos permutaciones, se pueden generar descendientes no válidos. Por lo que se requiere la reparación de las cadenas inválidas que se producen [20]. Algunos de los operadores más utilizados se presentan a continuación.

Cruce en orden (OX)

Para generar nuevos individuos con este operador se deben seguir los pasos:

1. Seleccionar de forma aleatoria dos subrutas P_A y P_B.
2. Generar un hijo copiando una subruta de P_A.
3. Borrar de P_B los valores que ya se encuentren en el hijo.
4. Los valores faltantes del hijo son los valores presentes en P_B.

La Fig. 3.3 muestra este procedimiento de forma gráfica.

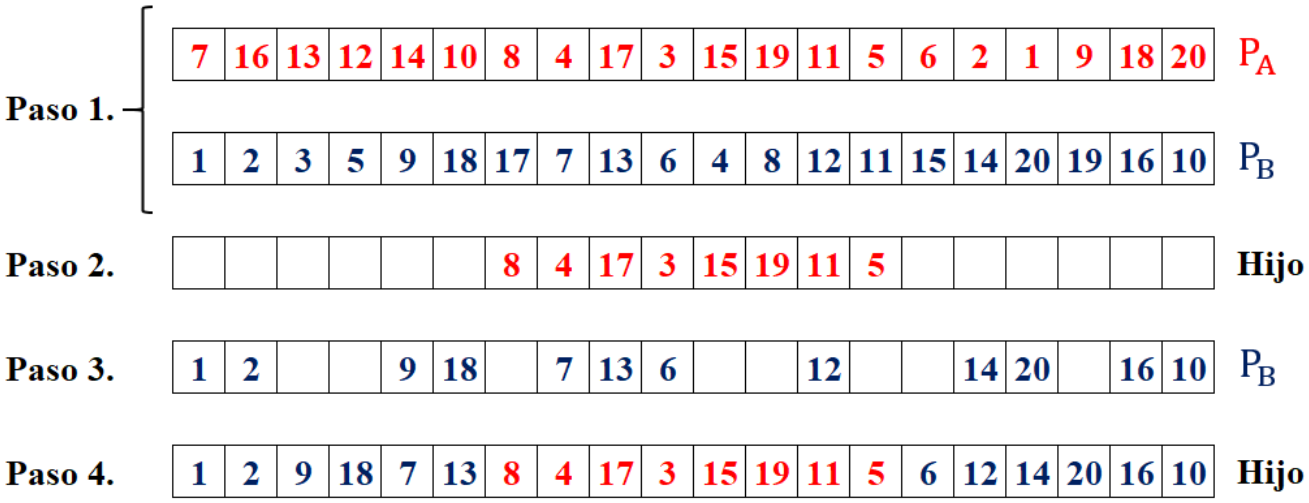


Figura 3.3: Cruce en orden (OX).

En las implementaciones clásicas de este operador se genera un hijo a partir de P_A y otro a partir de P_B, pero es posible generar una cantidad de individuos diferente según las necesidades de una implementación específica.

Cruce parcialmente mapeado (PMX)

Pasos para generar nuevos individuos con el operador PMX:

1. Seleccionar de forma aleatoria dos subrutas P_A y P_B.
2. Seleccionar de forma aleatoria dos puntos de cruce.
3. Intercambiar estos dos segmentos en los hijos que se generan.
4. El resto de las cadenas se obtienen haciendo mapeos entre los dos padres: a) Si un valor no está contenido en el segmento intercambiado, permanece igual; b) si está contenido se sustituye por el valor que tenga dicho segmento en el otro padre.
5. Finalmente se analizan los hijos generados para completar los valores faltantes en caso de que se presenten.

Un ejemplo del funcionamiento de este operador se muestra en la Fig. 3.4.

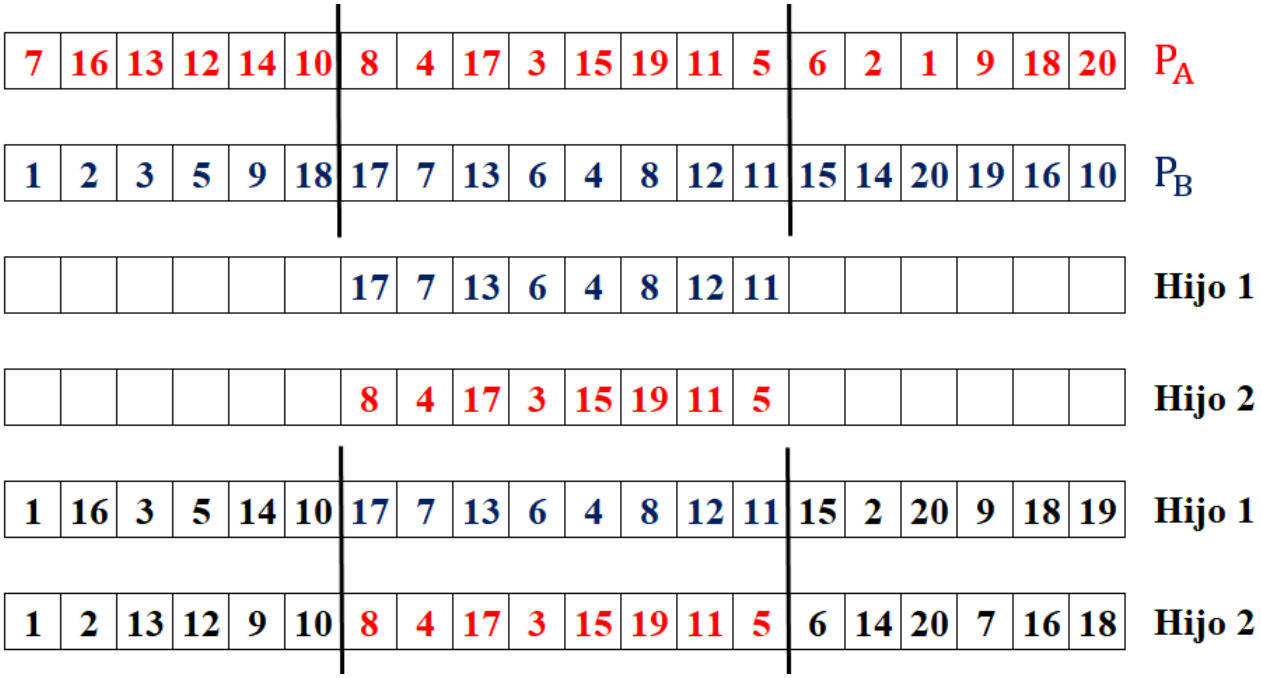


Figura 3.4: Cruce parcialmente mapeado (PMX).

Cruce basado en posición

El algoritmo de este operador es el siguiente:

1. Seleccionar de forma aleatoria dos subrutas P_A y P_B.

2. Seleccionar de forma aleatoria un conjunto de posiciones de P_A.

3. Generar un hijo borrando de P_A con los valores seleccionados en el paso anterior.

4. Borrar los valores seleccionados de P_B.

5. Colocar en el hijo los valores faltantes de acuerdo a la secuencia de P_B.

Este operador se describe de forma gráfica con el ejemplo de la Fig. 3.5.

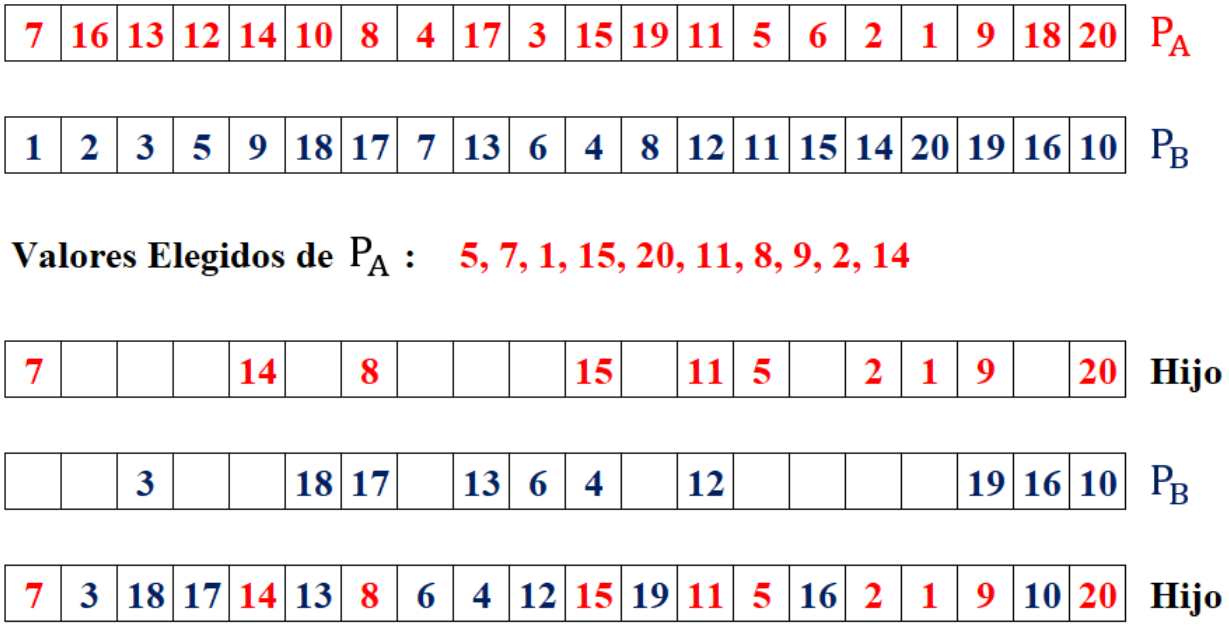


Figura 3.5: Cruce basado en posición.

Operador de cruce EAX

En el estudio del estado del arte para la solución del TSP se encontró el **operador de cruce con ensamble de aristas (EAX)** en [5], el cual genera buenos resultados reduciendo el costo computacional respecto al tiempo de ejecución de los AG para el TSP, y conservando la diversidad de las poblaciones de los AG.

El operador EAX una de las principales cualidades de esta investigación y será explicado con detalle al final de este capítulo.

3.2.2. Mutación para permutaciones

Una vez generado un nuevo individuo a partir de la operación de cruce, éste se modifica con una probabilidad previamente establecida. Los operadores de mutación clásicos para permuta-

ciones se describen a continuación.

Mutación por intercambio

La mutación por intercambio genera dos posiciones al azar dentro del rango de valores que conforman una ruta para después intercambiar los valores correspondientes a dichas posiciones. La Fig. 3.6 muestra de forma gráfica el funcionamiento de este operador.

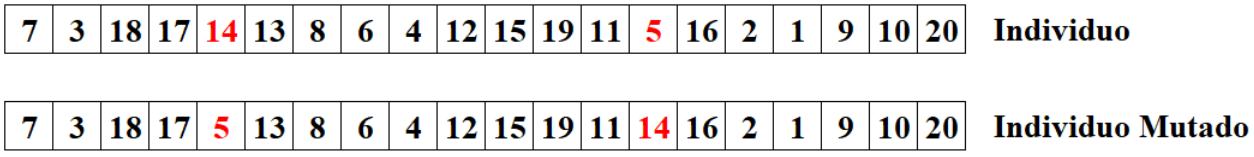


Figura 3.6: Mutación por intercambio.

Mutación por inversión

Este operador invierte la subruta acotada por dos posiciones generados de manera aleatoria. Este operador se describe de forma gráfica con el ejemplo de la Fig. 3.7.

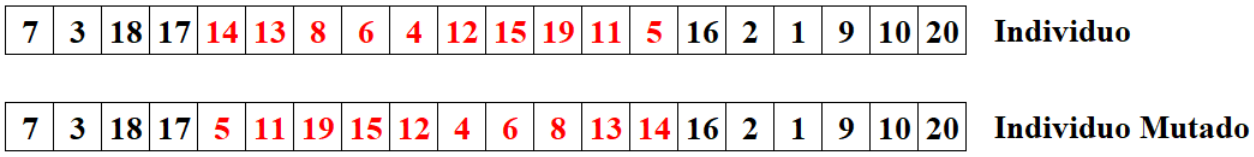


Figura 3.7: Mutación por inversión.

3.3. Algoritmos genéticos paralelos

Las AG son muy aptos para la paralelización dado que trabajan con conjuntos de individuos independientes; los operadores genéticos pueden ser paralelizados, por lo que es fácil distribuir la carga computacional entre múltiples procesadores [21]. Los programas paralelos se basan en la idea de dividir una tarea y resolver las subtareas de forma simultáneamente en múltiples unidades de procesamiento. Esta idea se puede aplicar a los AG de muchas formas, algunas de ellas utilizan una sola población, y otras dividen a la población en varias subpoblaciones aisladas [15]. Existen tres modelos principales de algoritmos genéticos paralelos (AGP), mismos que se describen a continuación.

3.3.1. Modelo maestro-esclavo

Este modelo aplica el operador de cruce a nivel global; es decir, los individuos pueden aparearse libremente en la población completa. Estos AGP trabajan con una población gestionada por un procesador maestro. La evaluación de la aptitud (calidad) de los individuos y la aplicación de los operadores genéticos es realizada por procesadores esclavos.

En la Fig. 3.8 se presenta un esquema gráfico de este modelo; donde cada color representa un conjunto de individuos y al procesador esclavo que aplicará los operadores genéticos. Los individuos conforman una única población global (circulo azul). El procesador maestro asigna los individuos que cada esclavo va a procesar.

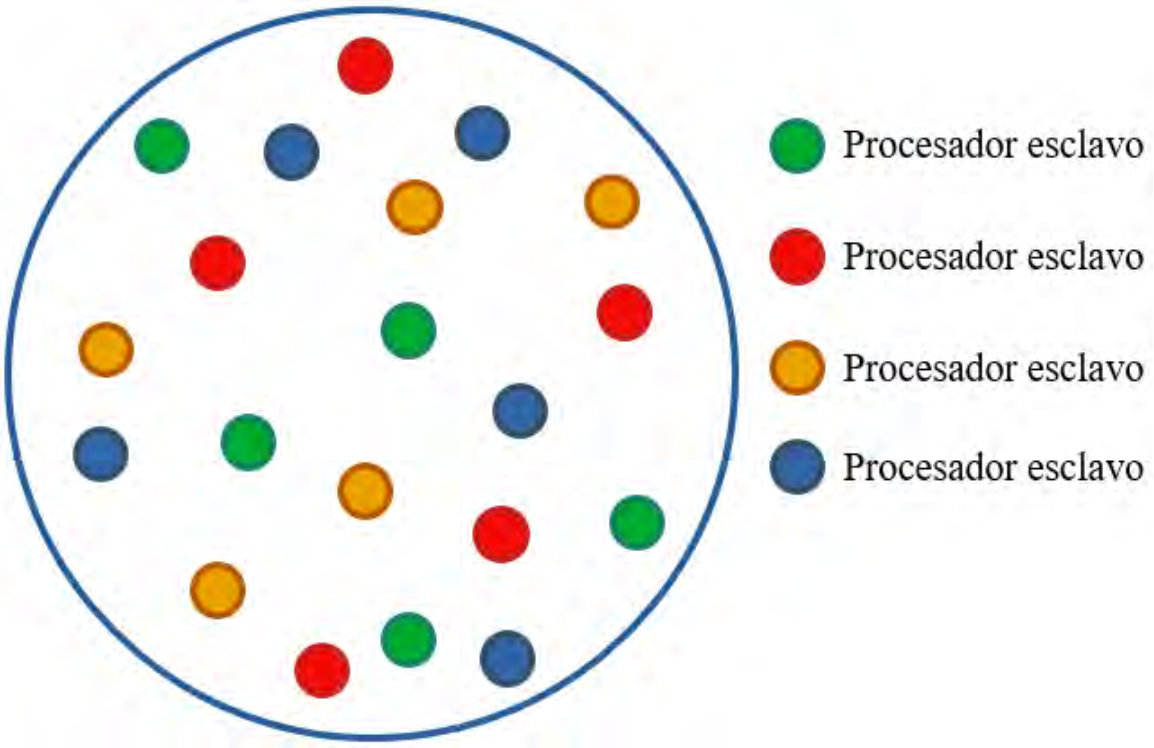


Figura 3.8: AGP con modelo maestro-esclavo.

En el intercambio de información el procesador maestro envía un subconjunto de individuos a cada procesador esclavo y estos devuelven los valores de aptitud para cada uno de ellos. La comunicación se puede implementar de manera síncrona o asíncrona. En la primera el procesador maestro espera a recibir la información de los individuos para generar la siguiente generación, mientras que en la segunda el AGP no espera a que todos los nodos envíen sus valores de aptitud.

Este tipo de algoritmos pueden implementarse sin problemas en computadoras con memoria compartida y con memoria distribuida. En computadoras de memoria compartida la población

podría estar en la memoria global y cada procesador esclavo deberá leer directamente los individuos previamente asignados. En memoria distribuida, el procesador maestro estaría encargado de distribuir individuos a los procesadores esclavos para recogerlos una vez evaluada la aptitud de cada uno de ellos.

3.3.2. Modelo celular

El modelo celular se ha diseñado para implementarse en computadoras masivamente paralelas. La población se divide entre los distintos procesadores y cada procesador debe albergar un único individuo. La selección y cruce se implementan entre individuos de un vecindario formado por individuos adyacentes. Se permite también el solapamiento entre vecindarios para permitir la interacción entre individuos de toda la población. El modelo celular se presenta de forma gráfica en la Fig. 3.9.

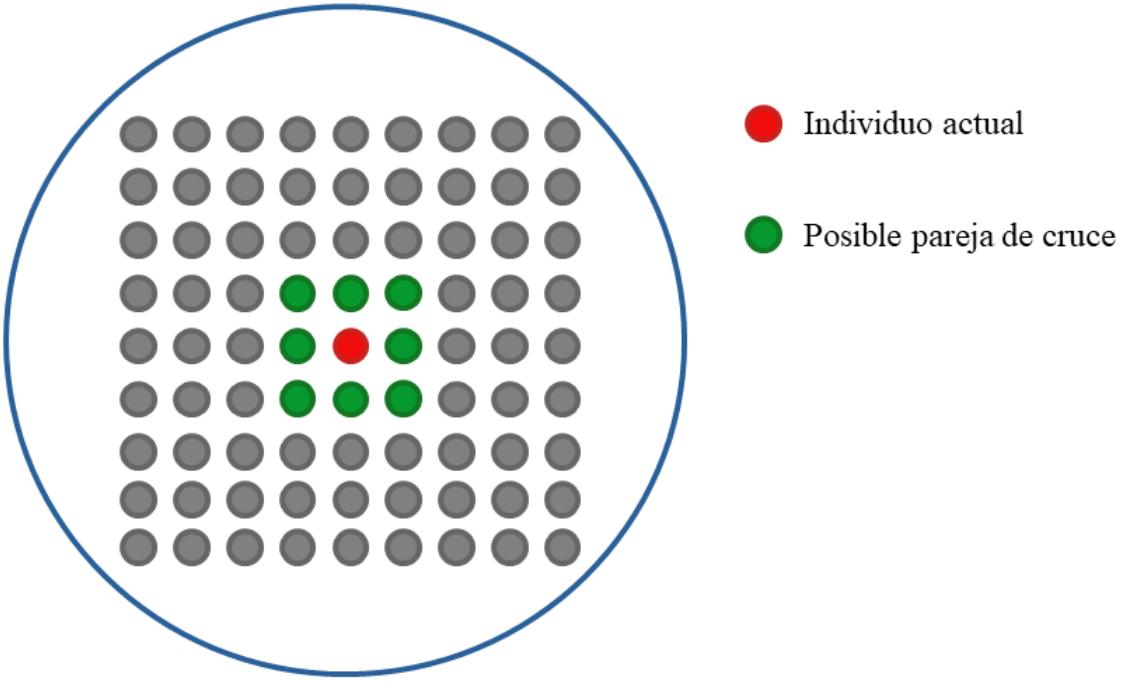


Figura 3.9: AGP con modelo celular.

Para seleccionar un individuo en una vecindad se escogen K individuos, y entre ellos se realizará una selección por torneo; esta selección puede ser determinista o probabilista. Este método es ideal en este modelo ya que no necesita ser implementado sobre toda la población global, puede ser implementado sobre subpoblaciones. El cruce se hace cambiando el individuo de una celda por el mejor de sus hijos, si este presenta mejor aptitud.

3.3.3. Modelo de islas

La principal característica de estos AGP son el uso de subpoblaciones con migración de individuos entre ellas. Debido a que cada una de las subpoblaciones evoluciona de manera independiente, la migración se vuelve muy importante para obtener buenos resultados. En la Fig. 3.10 se presenta un esquema gráfico de este modelo.

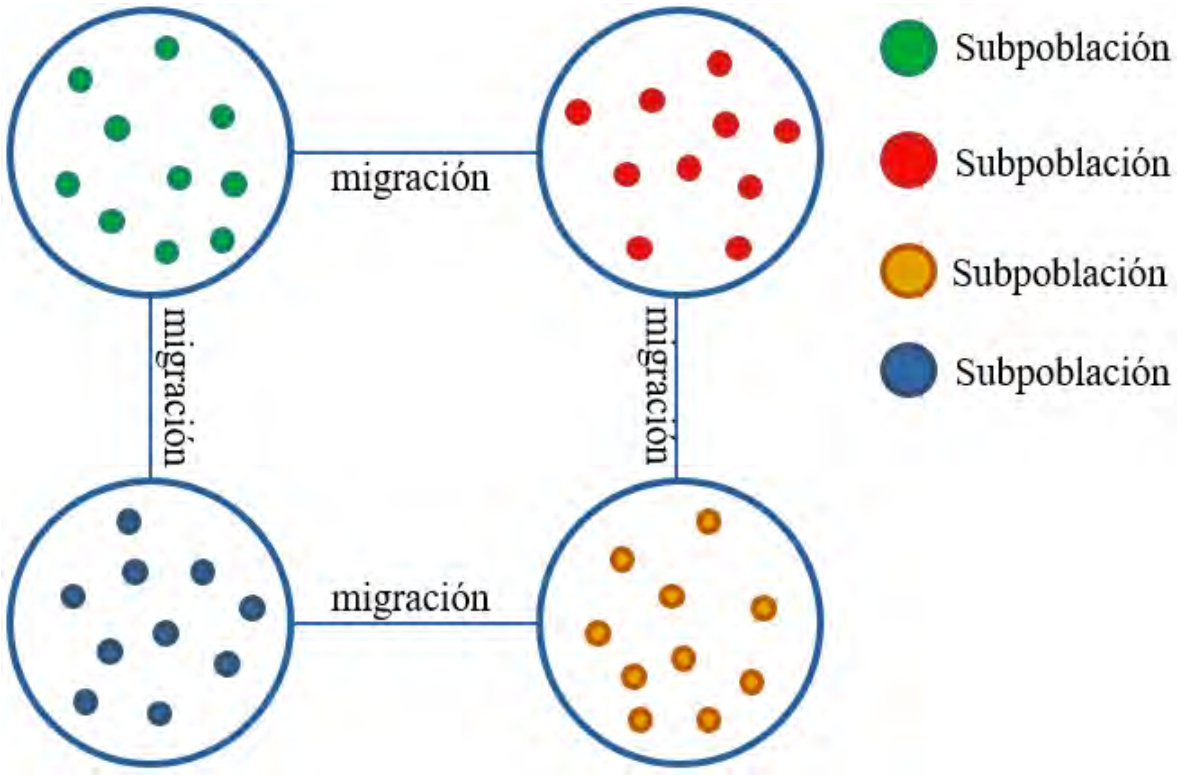


Figura 3.10: AGP con modelo de islas.

El buen rendimiento de estos algoritmos depende de varios factores, la frecuencia de las migraciones y la topología de comunicación entre subpoblaciones son las más importantes.

La migración prematura puede traer problemas debido a que la calidad de los individuos migrados podría no generar mejoras en las poblaciones; mientras que una migración tardía puede aumentar el tiempo de convergencia al óptimo global o propiciar que dicha convergencia no se produzca. Encontrar un momento adecuado para la migración es un aspecto muy importante a considerar en el diseño de este tipo de algoritmos.

La selección del individuo que migrará a otra población es también de suma importancia, algunos autores proponen migrar al mejor individuo de cada subpoblación a sus subpoblaciones adyacentes. Otra implementación se puede obtener al enviar al mejor individuo de cada subpoblación a un nodo maestro que se encargue de elegir a los mejores individuos recibidos y

de reenviarlos a todas las subpoblaciones. Generalmente el individuo que migra reemplaza al individuo con peor aptitud en la isla destino.

La topología de comunicación es muy importante para implementar un AGP con modelo de islas, esta determina la velocidad con la que se propagan buenas soluciones hacia el resto de las subpoblaciones. Si se tiene un alto grado de conectividad los buenos individuos se propagarán rápidamente a todas las subpoblaciones, favoreciendo la convergencia hacia un óptimo. En caso contrario, las subpoblaciones estarán más aisladas, evolucionando más despacio y tardando más tiempo en sumar genes buenos a las nuevas soluciones. Es necesario considerar que la densidad de conexiones supone una sobrecarga en el tráfico del canal de comunicación. Es de suma importancia la selección de una buena topología para obtener el máximo rendimiento del algoritmo.

3.4. Proceso de migración para el AGP con modelo de islas

La investigación realizada en [1] compara diferentes topologías de implementación de algoritmos genéticos paralelos con modelo de islas; haciendo un análisis estadístico de un conjunto de experimentos utilizando instancias de la biblioteca TSPLIB. A continuación, se presentan algunas de estas topologías.

3.4.1. Migración en topología de anillo

En esta topología las subpoblaciones o islas están distribuidas en anillo y la migración de individuos se hace de una isla a su isla vecina derecha o izquierda. En la Fig. 3.11 se presenta de forma gráfica esta implementación.

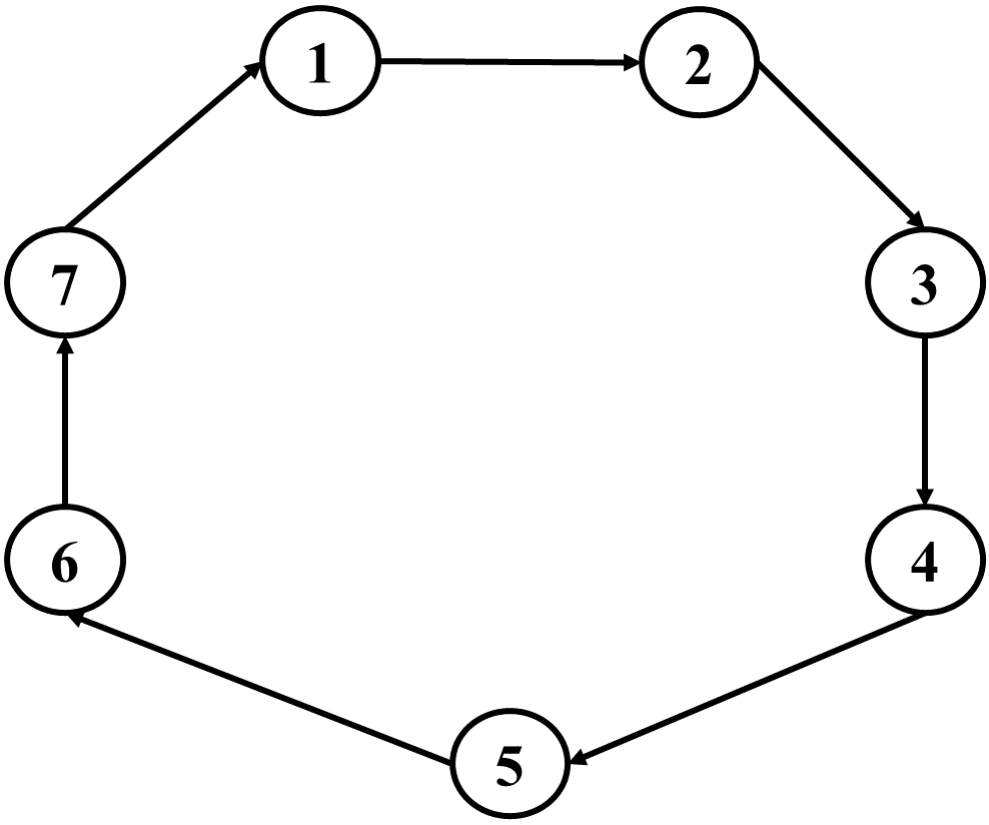


Figura 3.11: Migración en topología de anillo para siete islas.

3.4.2. Migración en topología de estrella

En la topología de estrella cada isla migra individuos a una misma isla, la cual se encarga de redirigir a los individuos recibidos a islas aleatorias diferentes de la isla de origen. Cuando los individuos migrados son los mejores de cada isla, la isla receptora suele denominarse isla élite ya que contiene a los mejores individuos de la población global. Una representación gráfica de la topología de estrella se indica en la Fig. 3.12.

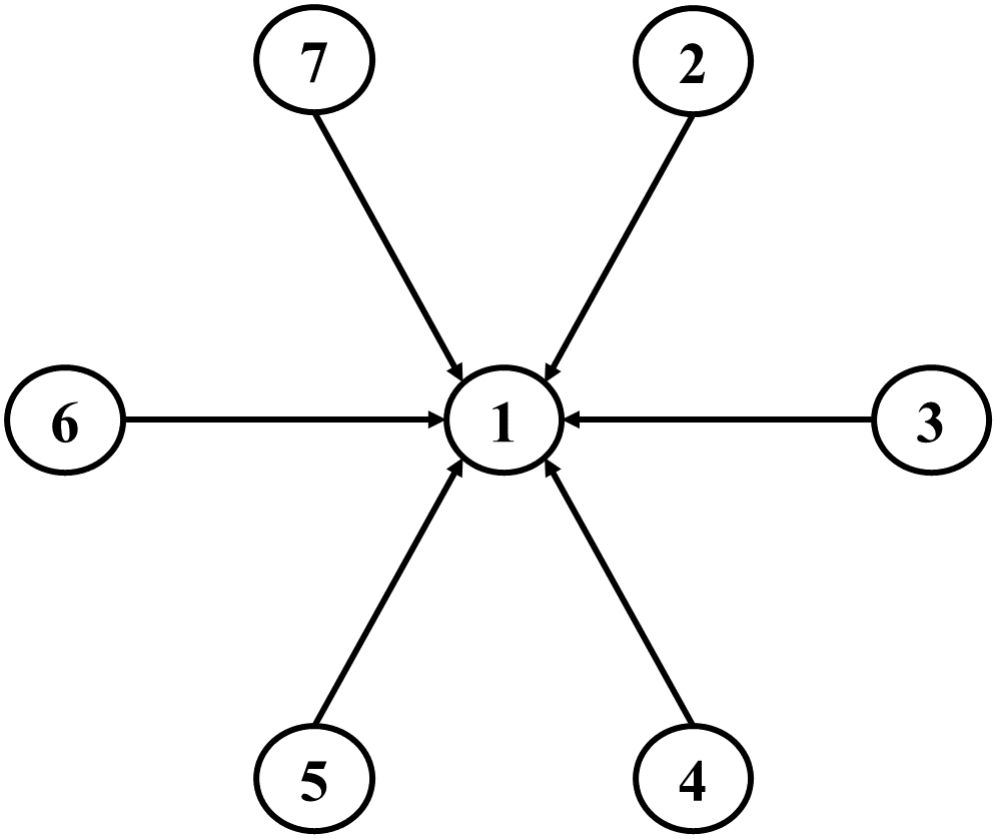


Figura 3.12: Migración en topología de estrella para siete islas.

3.4.3. Migración en topología de árbol ordenado

En este tipo de topología las islas se conectan en forma de árbol, una primera isla recibe individuos de otras islas conectadas y estas a su vez reciben individuos de islas que se encuentran en un tercer nivel. Aunque la isla primaria y las islas de niveles más profundo no tienen comunicación directa, dependiendo de los criterios de migración, estas islas pueden intercambiar individuos en un determinado momento. Esta topología se muestra de forma gráfica en la Fig. 3.13.

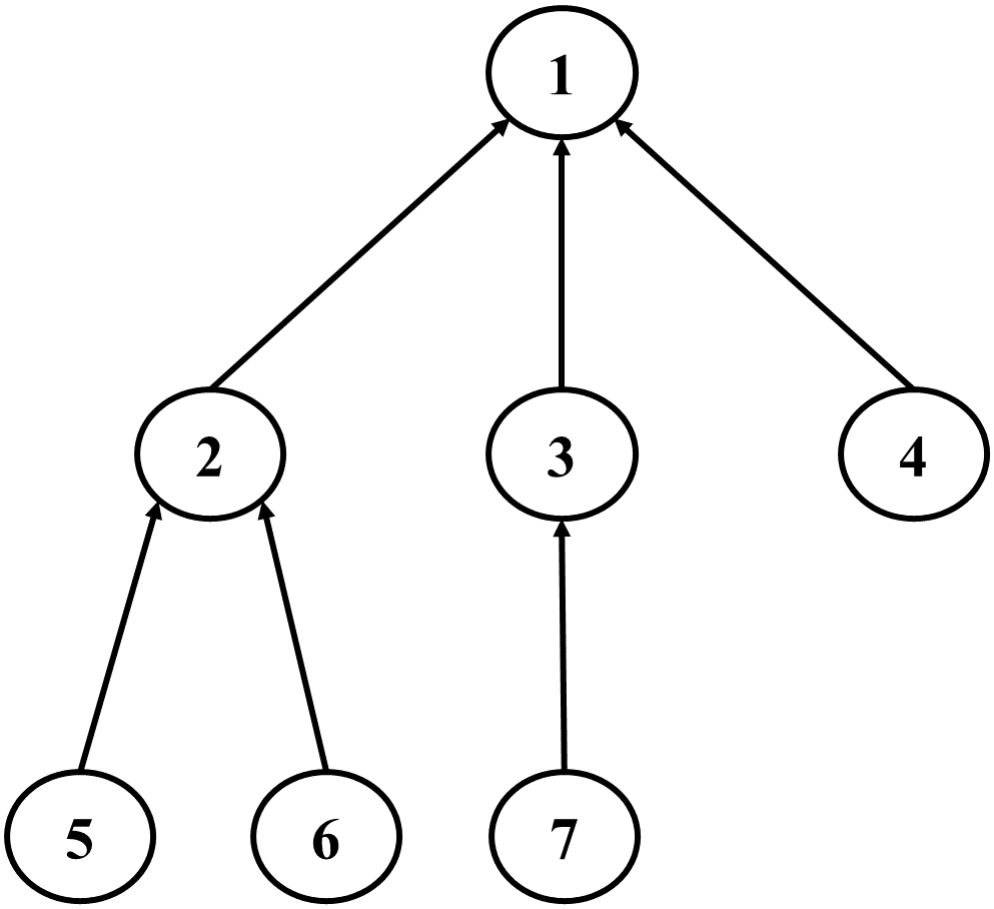


Figura 3.13: Migración en topología de árbol ordenado para siete islas.

3.4.4. Migración en topología de árbol binario completo

Esta topología comparte el mismo funcionamiento y propiedades que la topología de árbol ordenado, con la diferencia de que las islas están distribuidas como un árbol binario completo. La Fig. 3.14 es una representación de este tipo de topología.

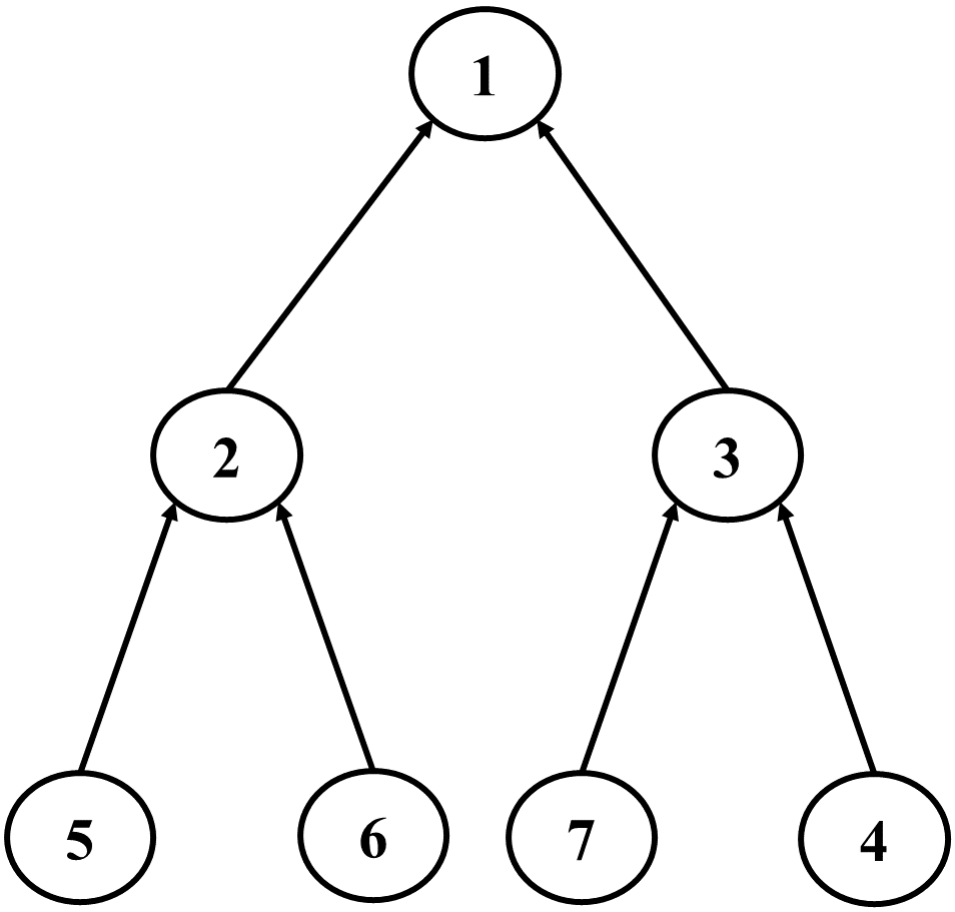


Figura 3.14: Migración en topología de árbol binario completo para siete islas.

3.5. El operador de cruce EAX

El operador EAX genera soluciones combinando nodos de dos soluciones padres y agregando pocos nodos nuevos que se determinan mediante un simple procedimiento de búsqueda. EAX tiene capacidades de optimización local y es uno de los operadores de cruce más efectivos en el estado del arte. Los operadores de búsqueda local evalúan una solución de manera más eficiente. Por ejemplo, una solución optimizada con el algoritmo 2-opt se puede obtener en tiempo $O(1)$.

El operador EAX genera soluciones a partir de una de las soluciones padres (P_A) al reemplazar pocos nodos con nodos seleccionados del otro padre (P_B). Esta implementación permite generar una solución en menos de $O(N)$ (N es la cantidad de nodos). EAX mantiene la diversidad de la población al implementarse con un algoritmo genético donde solo P_A se reemplaza con un individuo nuevo. Cuando EAX no puede generar una solución que mejore P_A, el número de nodos a reemplazar debe ser aumentado.

En un AG con EAX la población inicial debe ser generada mediante algún método de búsqueda local, siendo el algoritmo 2-opt el método implementado en este trabajo ya que es conveniente utilizar una búsqueda local simple para obtener soluciones de alta calidad minimizando el tiempo de ejecución requerido.

En cada generación del AG cada uno de los individuos es seleccionado como P_A y P_B una vez de manera aleatorio. Para cada par de padres P_A y P_B, EAX genera una determinada cantidad de descendientes, después selecciona al mejor de ellos y reemplaza a P_A si la calidad del descendiente seleccionado es mejor a la calidad de P_A. Por lo tanto, si todos los descendientes generados por P_A y P_B tienen una calidad menor a P_A, P_A no será reemplazado y formará parte de la siguiente generación. Esto inhibe la pérdida de diversidad en la población, por lo que no es necesario utilizar operadores de mutación.

En la implementación desarrollada en esta investigación se define una instancia del TSP como un grafo completo que consta de N nodos y una función d que devuelve la distancia entre cada par de nodos. Después se seleccionan dos individuos P_A y P_B como un par de padres para iniciar el proceso de cruce.

A continuación, se genera una cantidad determinada de subrutas, donde cada subruta se obtiene mediante el siguiente proceso: se selecciona un primer nodo aleatorio de P_A, el segundo nodo se obtiene de P_B como el nodo vecino izquierdo o derecho del nodo previamente seleccionado, este procedimiento se repite hasta completar una subruta de tamaño previamente definido y evitando repetir nodos.

Una implementación básica del operador EAX genera una única subruta de nodos alternados de P_A y P_B, este procedimiento es la principal diferencia entre EAX y otros operadores de cruce para el TSP; se puede decir que es el núcleo del operador. La subruta obtenida por el núcleo de EAX debe ser analizada y completada para obtener una nueva ruta válida para el problema. Para completar una subruta se pueden implementar diferentes heurísticas, como por ejemplo el algoritmo del vecino más cercano, una heurística de inserción o algún método que implemente la estrategia de Lin-Kernighan.

3.5.1. Versiones del EAX con diferentes heurísticas

Al implementar diferentes heurísticas para completar rutas válidas se hace evidente que el EAX puede derivar en diferentes operadores de cruce que compartirán el núcleo de EAX y diferirán en la heurística señalada anteriormente.

EAX con heurística del vecino más cercano

Una vez obtenida la subruta de nodos alternados generada por el núcleo de EAX; esta implementación trata de construir una ruta de bajo costo basándose en el nodo más cercano al último nodo establecido en la subruta, mismo que no debe estar repetido en la subruta generada hasta ese momento; esta lógica se aplica para buscar todos los nodos faltantes y generar una ruta completa válida para el problema.

La heurística del vecino más cercano busca moverse de un nodo al siguiente, de manera que, de todas las opciones, el nodo elegido sea el más cercana al nodo actual. Este algoritmo escoge la mejor opción que tiene disponible en una iteración sin ver que esto puede obligarle a tomar malas decisiones posteriormente. Al finalizar el proceso probablemente quedarán nodos cuya conexión obligará a introducir aristas de costos elevado.

EAX con heurística de inserción

Del mismo modo que el caso anterior, este método inicia con la subruta de nodos alternados. Posteriormente debe extender dichos nodos insertando los nodos restantes. En cada paso se inserta un nuevo nodo a la subruta hasta obtener una ruta válida para el problema.

EAX con algoritmo 2-Opt

Este algoritmo llena de manera aleatoria los nodos faltantes de la subruta de nodos alternados; después elimina dos aristas que se cruzan y reconecta los dos caminos resultantes mediante aristas que no se crucen, el ciclo final es más corto que el inicial.

EAX con algoritmo k-Opt

A partir de la subruta de nodos alternados, k-Opt divide en k partes en lugar de dos y combina los caminos resultantes de la mejor manera posible. Se le llama movimiento k-Opt a tal modificación. Al aumentar k aumentará el tamaño del entorno y el número de posibilidades de movimientos. Debido al número de combinaciones para eliminar k aristas en un ciclo, k-Opt sólo pueda ser aplicado a instancias del TSP de tamaño pequeño.

Capítulo 4

Implementación del AGPH-I con el operador EAX

La metodología par la implementación del algoritmo genético paralelo hibrido con modelo de islas se presenta en este capítulo.

La computación en paralelo se ha convertido en el paradigma dominante en el diseño de las nuevas arquitecturas de computadoras. Procesadores multinúcleo y multiprocesador, clústeres y procesadores paralelos masivos (MPP) son algunas clasificaciones de computadoras paralelas según el paralelismo que admite su hardware; en muchas ocasiones se combinan arquitecturas paralelas junto a procesadores tradicionales para obtener un mayor rendimiento al implementar algoritmos muy específicos.

La programación de software en paralelo tiene mayor dificultad que la programación secuencial ya que conlleva nuevas fuentes de errores como lo son las condiciones de carreras, mientras que la comunicación y sincronización representan grandes obstáculos para alcanzar un mayor rendimiento, también la depuración del código es mucho más compleja y pueden aparecer errores que no aparecen cuando se implementa en secuencial.

Un proceso es una instancia de un programa que se está ejecutando, cada instancia tiene su propio espacio de memoria y estado de ejecución. Un proceso tiene un espacio de memoria y tiene al menos un flujo de control llamado hilo. Cuando un programa corre, el valor del contador de programa determina la próxima instrucción a ejecutar. La secuencia de instrucciones resultante es llamada hilo de ejecución. Múltiples hilos evitan hacer cambios de contexto y permiten compartir código y datos. Debido a que entre varios hilos de un mismo proceso se comparte el mismo espacio de memoria del proceso; se requieren mecanismos de sincronización.

La implementación se llevó a cabo mediante hilos POSIX que es un estándar para la programación de multiprocesos basado en el lenguaje de programación C, mismos que son ideales para la paralelización de tareas en procesadores multinúcleos. Los hilos de un proceso comparten

un espacio de direcciones común, por lo que todos los hilos de un proceso pueden acceder a las variables globales. Cada subproceso tiene una pila de tiempo de ejecución separada que se utiliza para controlar las funciones activadas y almacenar sus variables locales. Estas variables declaradas localmente dentro de las funciones son datos locales del hilo en ejecución y otros hilos no pueden accederlas directamente, dado que la pila de tiempo de ejecución de un subproceso se elimina después de que se termina un subproceso.

Los tipos de datos, interfaz y macros del estándar de hilos POSIX (Pthreads) generalmente están disponibles a través del archivo de encabezado <pthread.h>. Este archivo de encabezado debe incluirse en un programa con hilos POSIX [22].

En la implementación obtenida se divide el total de la población del AG en la cantidad de hilos de ejecución definidos; a estas divisiones se les llama subpoblaciones y gracias a ellas se puede reducir el tiempo de ejecución necesario para encontrar una solución de buena calidad. Es decir, para un procesador de 4 núcleos se definen 4 hilos de ejecución y una población de 80 individuos obteniendo 4 subpoblaciones de 20 individuos cada uno.

La explicación de la implementación del algoritmo AGPH-I se presenta de dos maneras, mediante el diagrama de flujo de la Fig. 4.1 y mediante el pseudocódigo del Algoritmo 1.

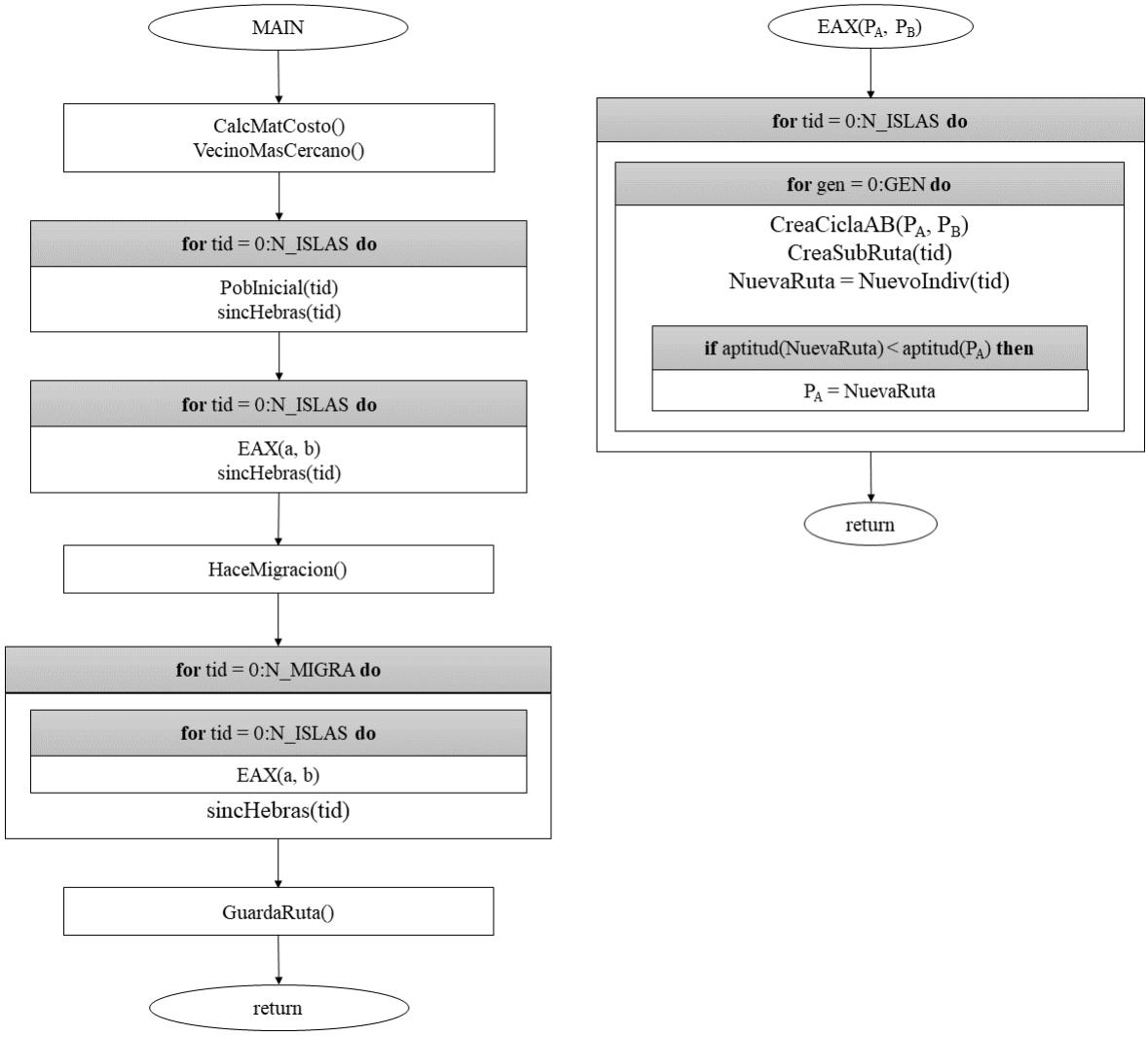


Figura 4.1: Diagrama de flujo representativo del algoritmo AGPH-I.

El algoritmo AGPH-I se describe en el pseudocódigo del Algoritmo 1, representa lo mismo que el diagrama de flujo mostrado en la Fig. 4.1.

Algoritmo 1 Pseudocódigo para el algoritmo AGPH-I

Entrada: *Una instancia del TSP*

Salida: *La ruta mínima*

```
1:  function MAIN
2:      CalcMatCosto()
3:      VecinoMasCercano()
4:      for tid=0:N_ISLAS do                          ▷ Se ejecuta en paralelo
5:          PobInicial(tid)
6:          sincHebras(tid)
7:      for tid=0:N_ISLAS do                          ▷ Se ejecuta en paralelo
8:          EAX(a,b)
9:          sincHebras(tid)
10:     HaceMigracion()
11:     for tid=0:N_MIGRA do
12:         for tid=0:N_ISLAS do                      ▷ Se ejecuta en paralelo
13:             EAX(a,b)
14:         sincHebras(tid)
15:     GuardaRuta
    return
16: function EAX(a, b)
17:     for tid=0:N_ISLAS do                          ▷ Se ejecuta en paralelo
18:         for gen=0:GEN do
19:             CreaCicloAB(P_A, P_B)
20:             CreaSubRuta(tid)
21:             NuevaRuta = NuevoIndiv(tid)
22:             if aptitud(NuevaRuta)<aptitud(P_A) then
23:                 P_A = NuevaRuta
    return
```

4.1. Matriz de costo y lista de vecinos cercanos

En la implementación desarrollada se utilizan dos estructuras de datos que son calculadas al inicio del algoritmo principal; dichas estructuras son la matriz de costos y una lista de vecinos cercanos para cada vértice del problema.

4.2. Generación de la población inicial

Para generar cada individuo en la población inicial del algoritmo AGPH-I, primero se calcula una ruta (individuo) totalmente aleatoria, la cual consta de una distancia de recorrido muy grande, es decir una función de aptitud muy alta. Una vez definida dicha ruta se aplica un método de búsqueda local (optimización) a la misma para mejorar significativamente la función de aptitud. El pseudocódigo del Algoritmo 2 define este proceso de optimización TOT_CIUDAD es la cantidad de nodos y POB_TOTAL es la cantidad de individuos presentes en la instancia del TSP seleccionada.

Algoritmo 2 Pseudocódigo para la generación de la población inicial

```
1:  function PobInicial(a, b)
2:      for i=0:POB_TOTAL do
3:          Ruta=CreaIndivAleatorio()
4:          while mejora==TRUE do
5:              mejora = FALSE
6:              for j=1:TOT_CIUDAD do
7:                  v1 = Ruta[j]
8:                  v2 = vecino[v1][i]
9:                  for k=1:PROF do
10:                     v3 = cercano[v1][k]
11:                     if pos[v2] < pos[v3] then
12:                         v4 = Ruta[pos[v3] + 1]
13:                         temp = d(v1, v2) + d(v3, v4) − (d(v1, v3) + d(v2, v4))
14:                         if max < temp then
15:                             max = temp
16:                             inversion()
17:                         mejora=TRUE
    return
```

Un ejemplo de este proceso de optimización se muestra en la Fig. 4.2, la cual corresponde a una instancia del TSP con 10 ciudades.

En este ejemplo, en la primera iteración (sección b de la Fig. 4.2) se asigna a v_1 la ciudad 1 y a v_2 la ciudad 2, que es el vecino siguiente de v_1; después se determina v_3 el vecino más cercano a v_1 que es el nodo 4, ahora se asigna en v_4 la ciudad a la derecha de v_3. Ahora se compara la distancia $d(v_1, v_2) + d(v_3, v_4)$ con $d(v_1, v_3) + d(v_2, v_4)$. Finalmente, se intercambian los vértices v_2 y v_3 para mejorar la ruta.

En la segunda iteración (sección c de la Fig. 4.2) se asigna $v_1 = 4$ y $v_2 = 3$ (vecino derecho de 4); después se determina el vecino cercano a v_1 y está dado por $v_3 = 2$, se determina el vecino a la derecha de v_3 que es $v_4 = 5$. después se intercambian los vértices v_2 y v_3 para mejorar la ruta.

Este proceso de análisis e intercambio de vértices se aplica a todos los elementos de la ruta, obteniendo una ruta libre de cruces de aristas y con un costo menor al costo inicial de la misma ruta.

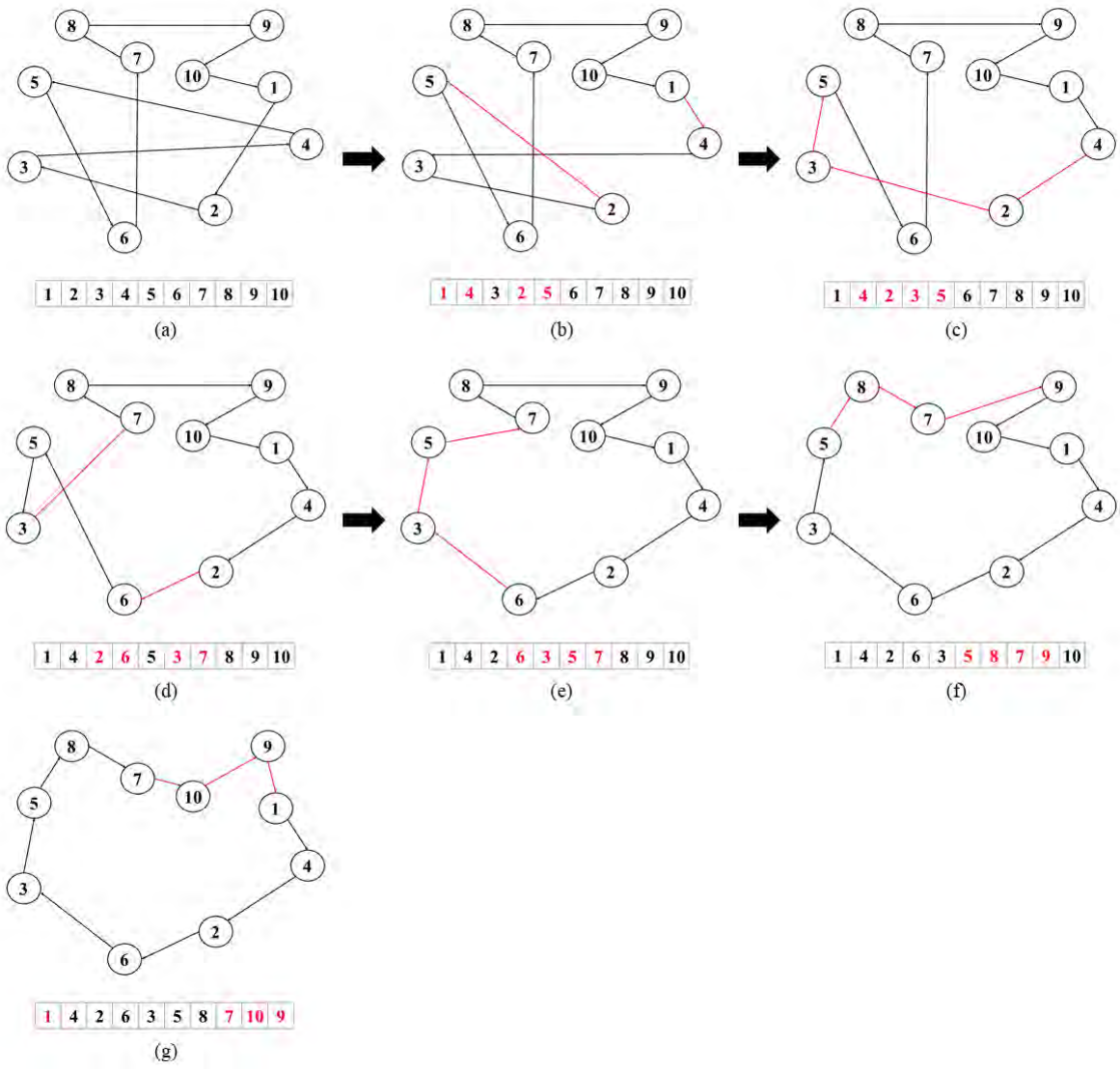


Figura 4.2: Ruta optimizada con el método 2-Opt.

Los pasos necesarios para implementar el operador EAX de forma general se describen en [5]. A continuación, se presenta dicha descripción para después detallar la implementación

desarrollada en este trabajo de investigación.

4.3. Descripción de los pasos del operador EAX

Para aplicar el operador de cruce EAX se define el TSP como un grafo completo (V, E) que consta de N vértices y una función d que devuelve la longitud (distancia) de cada arista. Se definen P_A y P_B como el par de padres con los que se va a realizar el cruce. También se definen E_A y E_B como el conjunto de aristas de P_A y P_B respectivamente. Finalmente, se define C_E como el conjunto de aristas que consiste en la solución de descendientes o solución intermedia.

Con el operador de cruce EAX se generan nuevos individuos a través de los siguientes cinco pasos.

1. Definir $G_{AB} = (V, E_A \cup E_B)$ como un multígrafo (grafo doble) no dirigido traslapando vértices de E_A y E_B.

2. Dividir las aristas de G_{AB} en ciclos AB (un ciclo AB se define como una subruta de G_{AB}) de modo que las aristas de E_A y de E_B estén unidos de forma alternada. Las aristas para el ciclos AB se selecionan de la siguiente manera: Seleccionar de forma aleatoria un vértice. Comenzando desde el vértice seleccionado, seleccionar las aristas de E_A y E_B en G_{AB} alternadamente hasta que se encuentre un ciclo AB, donde la arista se selecciona aleatoriamente (si existen dos candidatos).

3. Definir un conjunto E como la unión de ciclos AB seleccionados previamente de acuerdo a una estrategia de selección dada.

4. Generar una solución intermedia a partir de P_A eliminando las aristas de E_A y agregando las de E_B en el conjunto E. Una solución intermedia consiste en una o más subrutas y la solución final, es decir el nuevo individuo se obtiene al unir estas subrutas, esto se explica detalladamente en el paso 5.

5. Generar un descendiente (nuevo individuo) al conectar todos las subrutas en una ruta. Los detalles se indican en la subsección siguiente.

6. Si hay más descendientes por generar ir al paso 3. En caso contrario salir del ciclo.

4.3.1. Generar el nuevo individuo. Paso 5

En este paso el algoritmo genético se convierte en híbrido ya que la unión de las subrutas para generar el nuevo individuo se realiza mediante búsqueda local con un algoritmo 2-opt. Sea $S[v](v = 1, ...N)$ un arreglo donde todos los elementos se inicializan con cero.

5-1. Establecer $v_1, ..., v_k$ como uno de los extremos de los bordes de E_A en el conjunto E, cada uno de los cuales es uno de los extremos de cada borde con un mayor valor de Pos (arreglo que grada la posición de las aristas). Ordenar $Pos[v_1], ..., Pos[v_k]$ en orden creciente. Ahora se pueden obtener los valores de la posición inicial y final para los k segmentos de la siguiente manera: $(Pos[v_1], Pos[v_2]-1), ..., (Pos[v_k-1], Pos[v_k]-1), (Pos[v_k], Pos[v_1]-1)$. Para cada segmento, asignar un ID de segmento arbitrariamente y determinar los valores de las posiciones adyacentes de acuerdo con los bordes de E_B en el conjunto E y Pos. Al rastrear los segmentos de acuerdo con los valores de la posición inicial, la posición final y la posición adyacente se pueden clasificar los segmentos de acuerdo con los subgrupos a los que pertenecen. Después, los ID de subrutas se asignan a los segmentos arbitrariamente. Ahora se puede calcular fácilmente m y $A_l(l = 1, ..., M)$.

5-2. Establecer r como el índice de la subruta más pequeña. Establecer V como un conjunto de vértices en este subgrupo que se obtiene al rastrear este subgrupo de acuerdo con $Link_C$, comenzando por un vértice incluido en él (almacenar dicho vértice para cada subgrupo). Establecer $S[v] = 1(v \in V)$.

5-3. Encontrar cuatro tuplas de aristas $e*, e'*, e''*, e'''*$, donde todos los pares de aristas $e \in U$ y $e' \in E_C \setminus U$ se generan de la siguiente manera. Sean e, e', e'' y e''' representados como $(v1, v2), (v3, v4), (v1, v3)$ y $(v2, v4)$, respectivamente.

$v_1 \in \{v | v \in V\}$,

$v_2 \in \{Link_C[v_1][0], Link_C[v_1][1]\}$,

$v_3 \in \{v | v \in Cercano(v_1, N), S[v] \neq 1\}$,

$v_4 \in \{Link_C[v_3][0], Link_C[v_3][1]\}$.

$Cercano(v_1, N)$ se refiere a un conjunto de vértices que se encuentran entre los N más cercanos a v_1.

5-4. Actualizar $Link_C$ de acuerdo con $E_C := (E_C \setminus \{e*, e'*\}) \cup \{e''*, e'''*\}$.

5-5. Establecer v_3* como un extremo del borde $e'*$ y encontrar el segmento que lo incluye, r' se obtiene como el ID de la subruta de ese segmento. Restar 1 de m, reasignar los ID de la subruta apropiadamente y recalcular $A_l(l = 1, ..., M)$ de acuerdo con la solución intermedia resultante. Si m es igual a 1 finalizar el procedimiento, de lo contrario ir al paso 5-2.

* Se debe tener en cuenta que la representación de segmento se usa solo para calcular m y $A_l(l = 1, ..., M)$ en el Paso 5-1 y nunca se actualiza, excepto para el ID de la subruta en los pasos 5-2 a 5-5. En el Paso 5-5, podemos recalcular fácilmente $A_l(l = 1, ..., M)$.

Para aplicar el operador de cruce EAX se genera un ciclo AB aplicando el pseudocódigo del Algoritmo 3.

Algoritmo 3 Generación del ciclo AB

```
1: function CreaCicloAB(P_A, P_B)
2:     largo = DIMENSION, H[0] = random % DIMENSION
3:     cicloAB[0] = P_A[0]
4:     for i=1:largo do
5:         lado = random % 2
6:         if i %2 = 1 then
7:             if SiguienteNoVisitado(H[i-1], P_A, lado) then
8:                 siguiente = SiguienteNoVisitado(H[i-1], P_A, lado)
9:             else if SiguienteNoVisitado(H[i-1], P_A, !lado) then
10:                siguiente = SiguienteNoVisitado(H[i-1], P_A, !lado)
11:            else
12:                siguiente = -1
13:        else
14:            if SiguienteNoVisitado(H[i-1], P_B, lado) then
15:                siguiente = SiguienteNoVisitado(H[i-1], P_B, lado)
16:            else if SiguienteNoVisitado(H[i-1], P_B, !lado) then
17:                siguiente = SiguienteNoVisitado(H[i-1], P_B, !lado)
18:            else
19:                siguiente = -1
    return
```

4.4. Pasos del operador EAX con un ejemplo

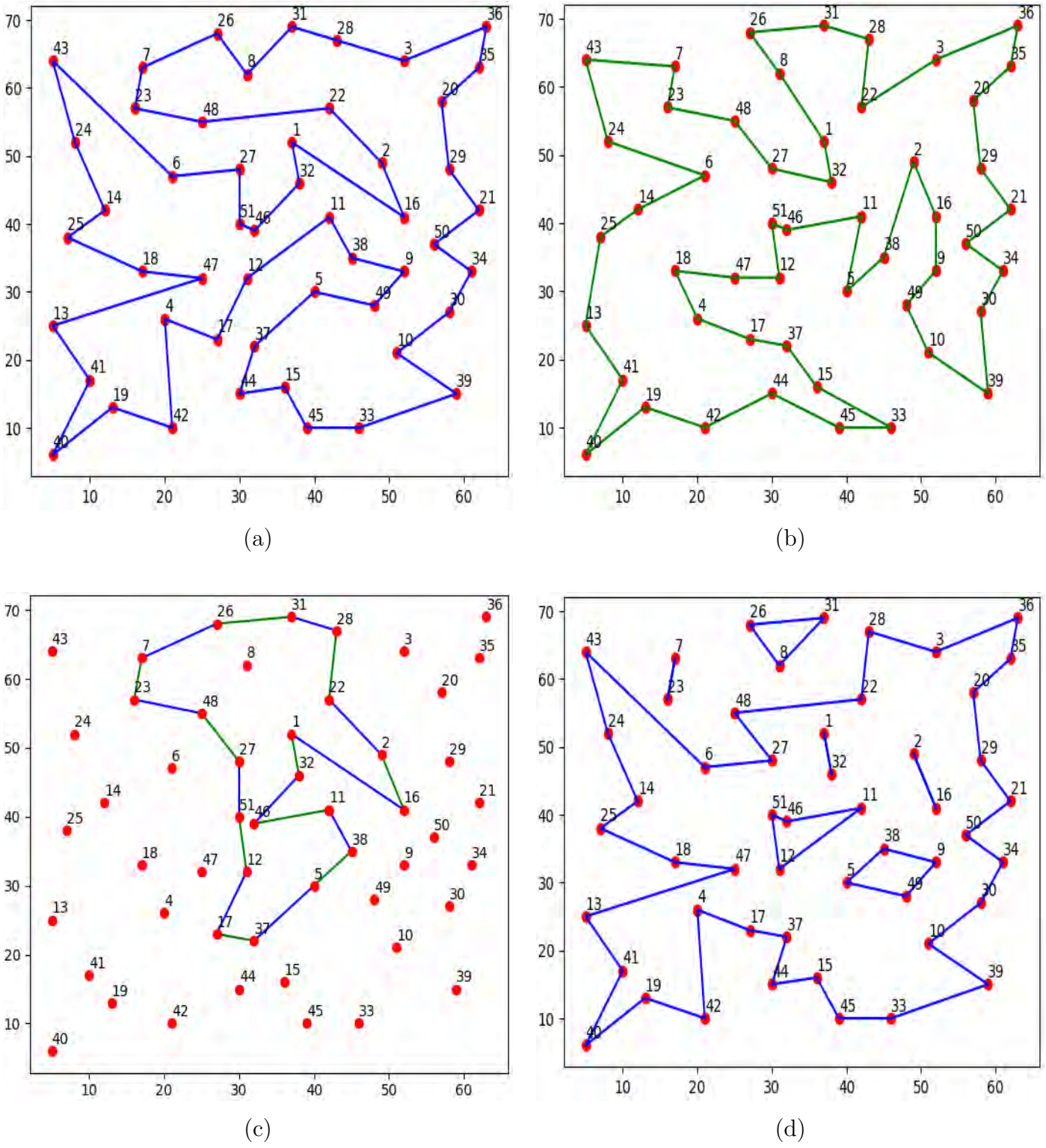
En la implementación de algoritmos genéticos existen diferentes métodos para seleccionar a los padres que generarán nuevos individuos, estrategias como la selección por torneo y selección por ruleta son métodos clásicos muy utilizados en la implementación de los AG.

En la implementación desarrollada en esta investigación se optó por utilizar un método que ha dado buenos resultados en [5] y [6]. Este consiste en seleccionar cada uno de los individuos de la población como P_A junto a otro individuo totalmente aleatorio como P_B en cada generación.

Es decir, todos y cada uno de los individuos tendrá la oportunidad de generar nuevos individuos y así podrá ayudar a la evolución de toda la población.

En la Fig. 4.3, (a) y (b) son dos individuos que generarán nuevas soluciones a partir del operador de cruce EAX, (c) es un ciclo AB generado por P_A (a) y P_B (b), y (d) son las subrutas generadas a partir del ciclo AB y P_A.

Los pasos para aplica el operador EAX se muestran en la Fig. 4.3 para la instancia eil51.tsp. Se toman dos individuos de la población inicial, representados en las figuras 4.3(a) y 4.3(b) que corresponden a los padres P_A y P_B respectivamente, al traslapar estas dos figuras se forma el multígrafo G_{AB}. A partir del multígrafo se forma un ciclo AB indicado en la Fig. 4.3(c). Posteriormente, quitando los vértices de P_A y dejando los vértices de P_B en el ciclo AB, que se unen con P_A se forman las subrutas de la Fig. 4.3(d).



(a) (b) (c) (d)

Figura 4.3: Proceso EAX (a) Ruta P_A, (b) Ruta P_B, (c) Ciclo AB generado de P_A y P_B y (d) Subrutas generadas del ciclo AB y P_A.

4.4.1. Obtención de un ciclo AB

El ciclo AB obtenido al aplicar el segundo paso del operador EAX se despliega en la Fig. 4.4. Este ciclo inicia en el vértice 28 que se seleccionó al azar de P_A.

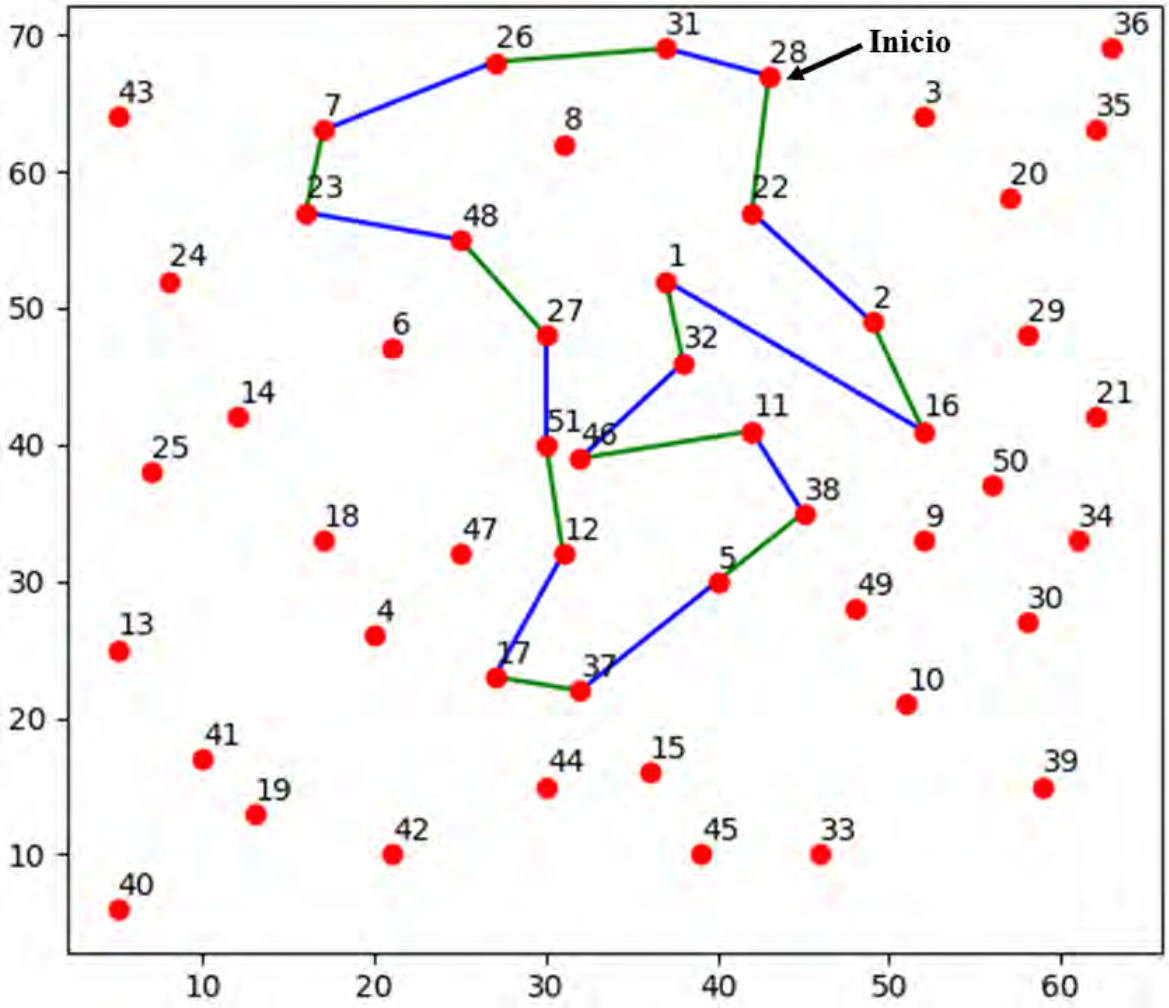


Figura 4.4: EAX: Obtención de un ciclo AB, arista verde pertenece a P_A y azul a P_B

4.4.2. Generación de subrutas

Una solución intermedia consta de un conjunto de subrutas. En la Fig. 4.5 se detalla la segmentación obtenida para una ruta solución de una instancia del TSP con 51 ciudades. Cada una de las subrutas generadas se resalta con colores diferentes y se describe como subruta 1, subruta 2, etc.

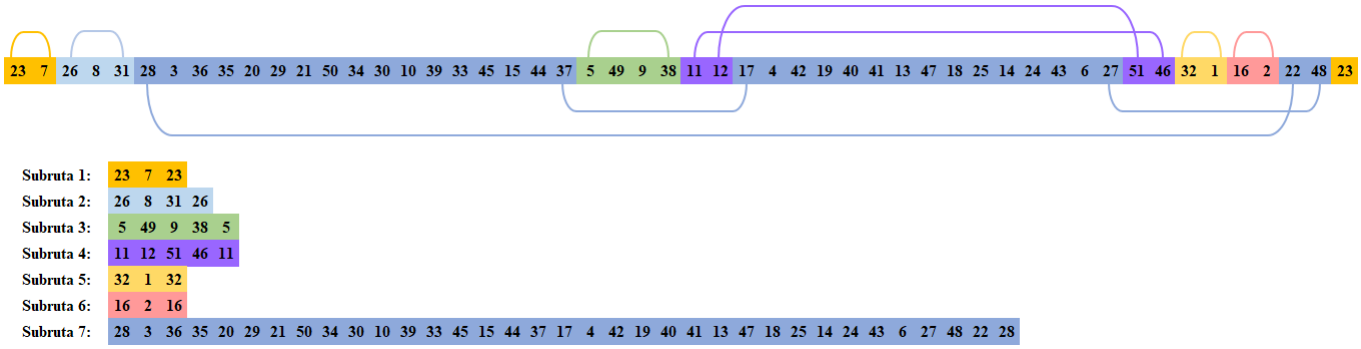


Figura 4.5: EAX: Segmentación de ruta.

En la Fig. 4.6 se presenta de forma gráfica los subrutas generadas a partir de la segmentación realizada.

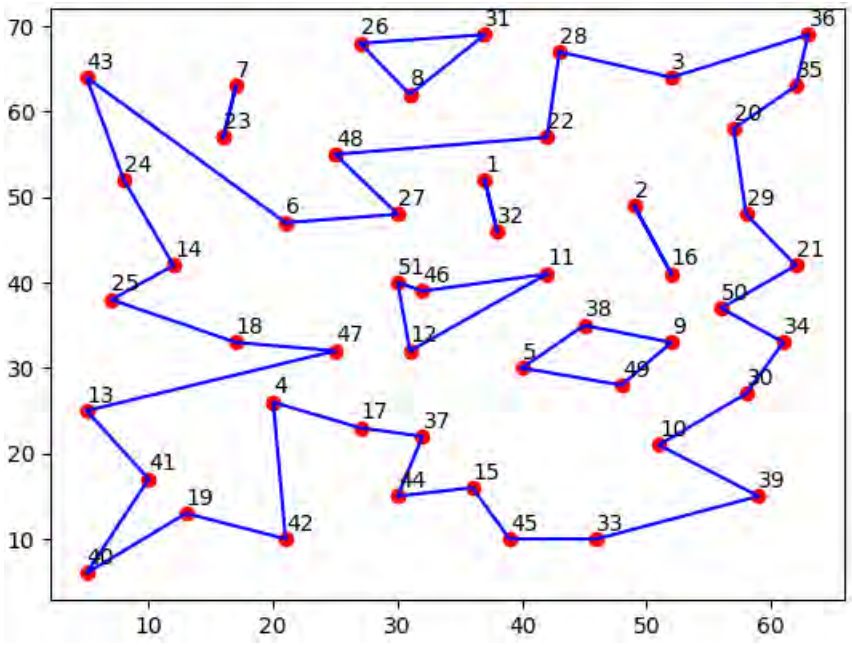


Figura 4.6: EAX: Subrutas generadas a partir de la segmentación de la Fig. 4.5.

4.4.3. Conectar las subrutas

Aplicando el método de búsqueda local, el algoritmo AGPH une las subrutas para generar una nueva ruta válida como posible solución a la instancia correspondiente.

En la Fig. 4.7 se muestra de forma gráfica la segmentación descrita por la Fig. 4.5 junto a las aristas que serán agregadas para optimizar la unión de las subrutas (líneas verdes) y las aristas que se han de eliminar (líneas rojas).

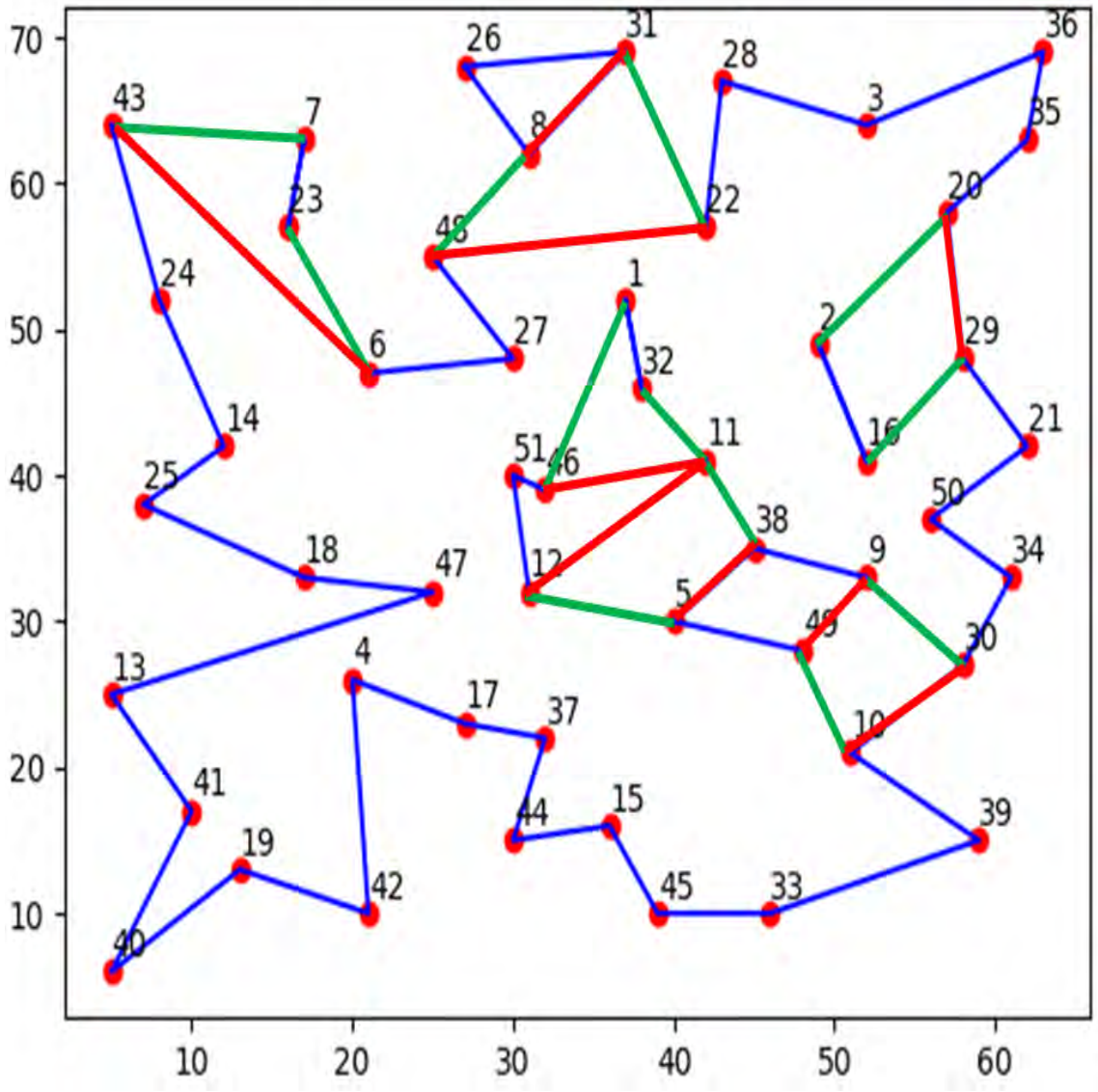


Figura 4.7: EAX: Unión de subrutas.

Finalmente, en la Fig. 4.8 se ilustra una nueva solución. Si la calidad de la nueva solución es mayor a la calidad de la solución en P_A, la nueva solución reemplazará a la solución de P_A.

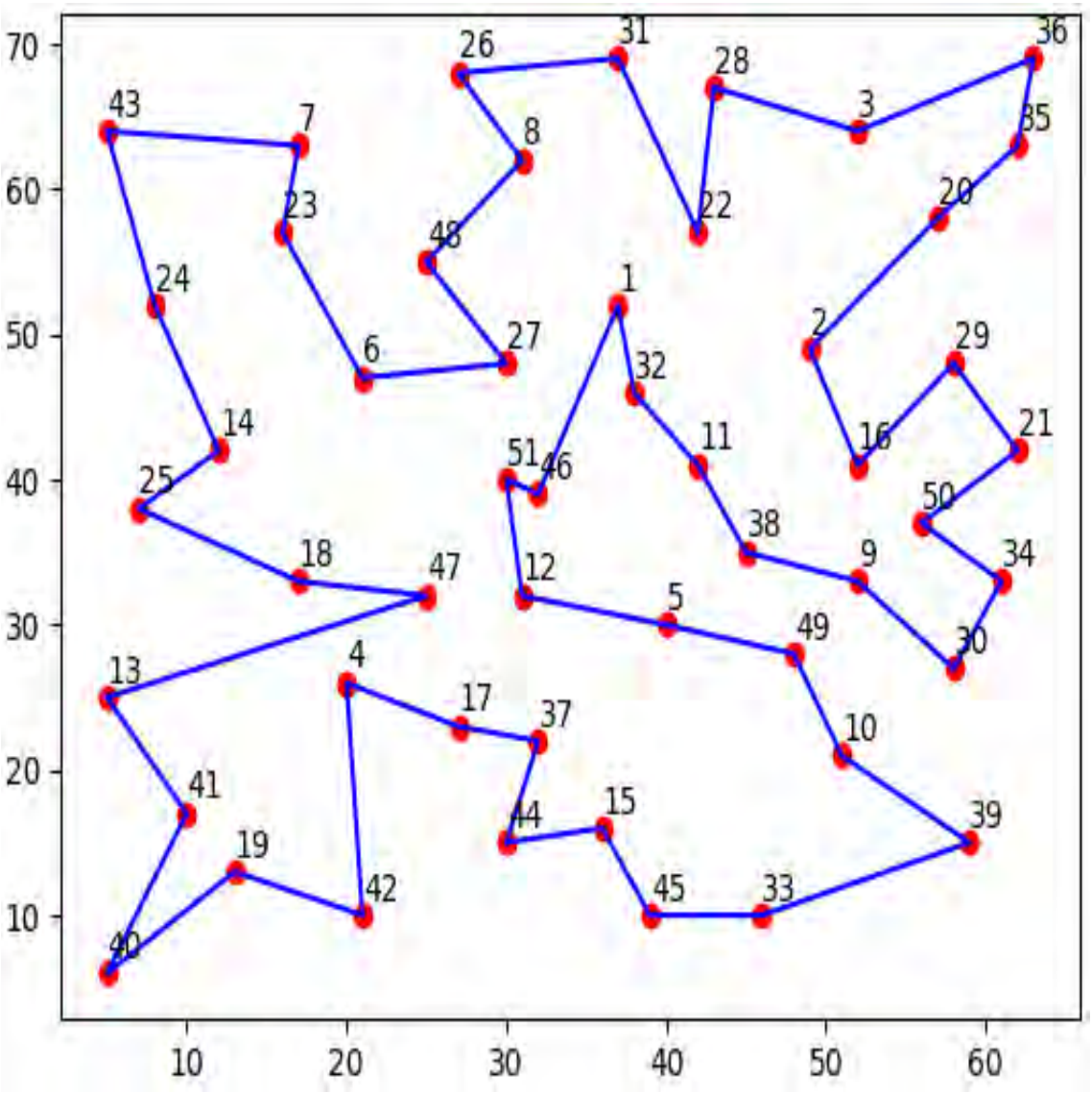


Figura 4.8: EAX: Ruta nueva.

4.5. Paralelización del algoritmo AGPH-I en un procesador multinúcleo

En el cómputo paralelo muchas instrucciones se ejecutan simultáneamente. Existen diversas formas de hacer cómputo paralelo (a nivel de bit, a nivel de instrucción, de datos y de tareas). La computación en paralelo se ha convertido en el paradigma dominante en la arquitectura de computadores. Procesadores multinúcleo y multiprocesador, clústeres y procesadores paralelos masivos (MPP) son algunas clasificaciones de computadoras paralelas según el parale-

lismo que admite su hardware; en muchas ocasiones se combinan arquitecturas paralelas junto a procesadores tradicionales para obtener un mayor rendimiento al implementar algoritmos muy específicos. La programación de software en paralelo tiene mayor dificultad que la programación secuencial ya que conlleva nuevas fuentes de errores como lo son las condiciones de carreras, mientras que la comunicación y sincronización representan grandes obstáculos para alcanzar un mayor rendimiento.

Los hilos POSIX es un estándar para la programación de multiprocesos basado en el lenguaje de programación C, mismos que son ideales para la paralelización de tareas en procesadores multinúcleo. Los hilos de un proceso comparten un espacio de direcciones común, por lo que todos los hilos de un proceso pueden acceder a las variables globales y a los objetos de datos generados dinámicamente. Cada subproceso tiene una pila de tiempo de ejecución separada que se utiliza para controlar las funciones activadas y almacenar sus variables locales. Estas variables declaradas localmente dentro de las funciones son datos locales del hilo en ejecución y otros hilos no pueden acceder directamente.

4.5.1. Migración y topología implementados

A partir de un numero de generaciones definido previamente, el AGP ejecuta un proceso de migración en la población total. La población total es dividida en una cantidad de subpoblaciones o islas que corresponde a la cantidad de subprocesos definidos en el AGP; cada una de estas islas es procesada por un subproceso.

Para iniciar el proceso de migración el número de generaciones debe ser mayor a un valor previamente definido, el cual es llamado *GEN*, a partir de que se cumpla esta condición el AGP realizará migración entre las islas.

Cada hilo selecciona al mejor individuo dentro de su isla y lo envía a otra isla aleatoria. Este proceso tiene el fin de aumentar el nivel de diversidad de la población total para evitar que cada isla se estanque en un óptimo local; esto implica que el AGP puede ser clasificado como modelo de islas, mismo que se ilustra de forma gráfica en la Fig. 3.10.

Capítulo 5

Resultados y discusión

En este capítulo se presentan los resultados obtenidos en los experimentos realizados. El objetivo de dichos experimentos es comparar diferentes implementaciones del algoritmo desarrollado, así como hacer una comparación con la implementación obtenida de [23]. Para lograr estas comparaciones, se ejecutaron las diferentes implementaciones resolviendo las mismas instancias del TSP y registrando los resultados en cada ejecución (nombre de la instancia, tamaño de población, error, tiempo de ejecución, entre otros).

El algoritmo desarrollado es implementado en una máquina virtual con 20 núcleos y 64 GBytes de memoria, bajo el sistema operativo Ubuntu 18.04. Las instancias del TSP utilizados en esta investigación se tomaron de la biblioteca [24], mismas que son del tipo TSP geométrico donde las coordenadas de cada ciudad se leen de un archivo de texto plano; y la distancia entre cada par de ciudades se calcula de la siguiente manera

$$d(v_i, v_j) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}, \tag{5.1}$$

donde $v_i(x_i, y_i)$ y $v_j(x_j, y_j)$ corresponden a las coordenadas de las ciudades v_i y v_j.

5.1. Descripción de los parámetros

Cuando se trabaja con algoritmos genéticos se manejan parámetros que indican el comportamiento de la convergencia para obtener la solución óptima. En este trabajo los parámetros son:

- Cantidad de islas, que nos define la cantidad de paralelismo, para cada isla se crea un hilo de ejecución.
- Tamaño de la población en cada isla.

- Cantidad de generaciones antes de iniciar la migración entre islas.
- Cantidad de iteraciones entre migración.
- Cantidad de veces que se hará migración, este parámetro funciona como un criterio de paro.
- Cantidad de descendientes que se evalúan cuando se aplica el operador de cruce EAX en cada par de individuos.
- Profundidad en la búsqueda local.

Estos parámetros se incluyen en las tablas de resultados, y constituyen los campos de las tablas como se indica a continuación:

- **PROB** es el nombre de la instancia de la biblioteca TSPLIB
- **POB** es la cantidad de individuos en la población global
- **ISLAS** es la cantidad de subpoblaciones o islas
- **GEN** es la cantidad de generaciones
- **OLIB** es el óptimo reportado en la biblioteca TSPLIB
- **Min** es el valor mínimo de las soluciones
- **Max** es el valor máximo de las soluciones
- **P** es el promedio de las soluciones
- **E %** es el porcentaje de error de Min con respecto a OLIB y está dado por

$$E(\,\%) = \frac{Min - Olib}{Olib} \times 100 \tag{5.2}$$

- **T seg** es el tiempo medido en segundos

5.2. Cantidad de memoria necesaria

Para almacenar las distancias entre la ciudades se utiliza una matriz de tamaño $N \times N$, el requerimiento de memoria crece como $\mathcal{O}(N^2)$, para guardar los vecinos más cercanos se utiliza una matriz de tamaño $N \times M$, con M la cantidad de vecinos más cercanos, como $N > M$. La instancia rbz43748.tsp requiere 7.13 GB y sra104815.tsp 40.93 GB

5.3. Generación de la población inicial

En el algoritmo AGPH-I se implementó la búsqueda local de una manera eficiente. En la Tabla 5.1 se muestra la comparación del tiempo de ejecución del algoritmo AGPH-I y el algoritmo AGPH-O descargado de la plataforma github [23], para una población de 128 individuos y una profundidad de vecinos más cercanos de 40. Se observa que nuestra implementación es mucho mejor. El factor de mejora es de 80.8 para 662 ciudades y a medida que se incrementa el tamaño del problema, el factor de mejora también se incrementa obteniéndose 268 para 2566 ciudades. También se observa que todos los valores de Min, Max y P son mejores en nuestra implementación.

Tabla 5.1: Generación población inicial con 128 individuos y profundidad de 40.

PROB	**OLIB**	**AGPH-I**				**AGPH-O**			
		Min	**Max**	**P**	**T seg**	**Min**	**Max**	**P**	**T seg**
xql662	2513	2720	2917	2804	0.26	2733	2947	2839	21
dkg813	3199	3443	3729	3559	0.35	3463	3765	3605	38
pbd984	2797	3035	3237	3128	0.44	3082	3290	3175	54
xit1083	3558	3826	4131	3989	0.52	3914	4181	4061	75
pds2566	7643	8346	8737	8545	2	8546	8942	8740	536

En la Fig. 5.1 se muestra de forma gráfica la diferencia en tiempo de ejecución del algoritmo AGPH-I y la implementación del algoritmo AGPH-O de la plataforma github [23].

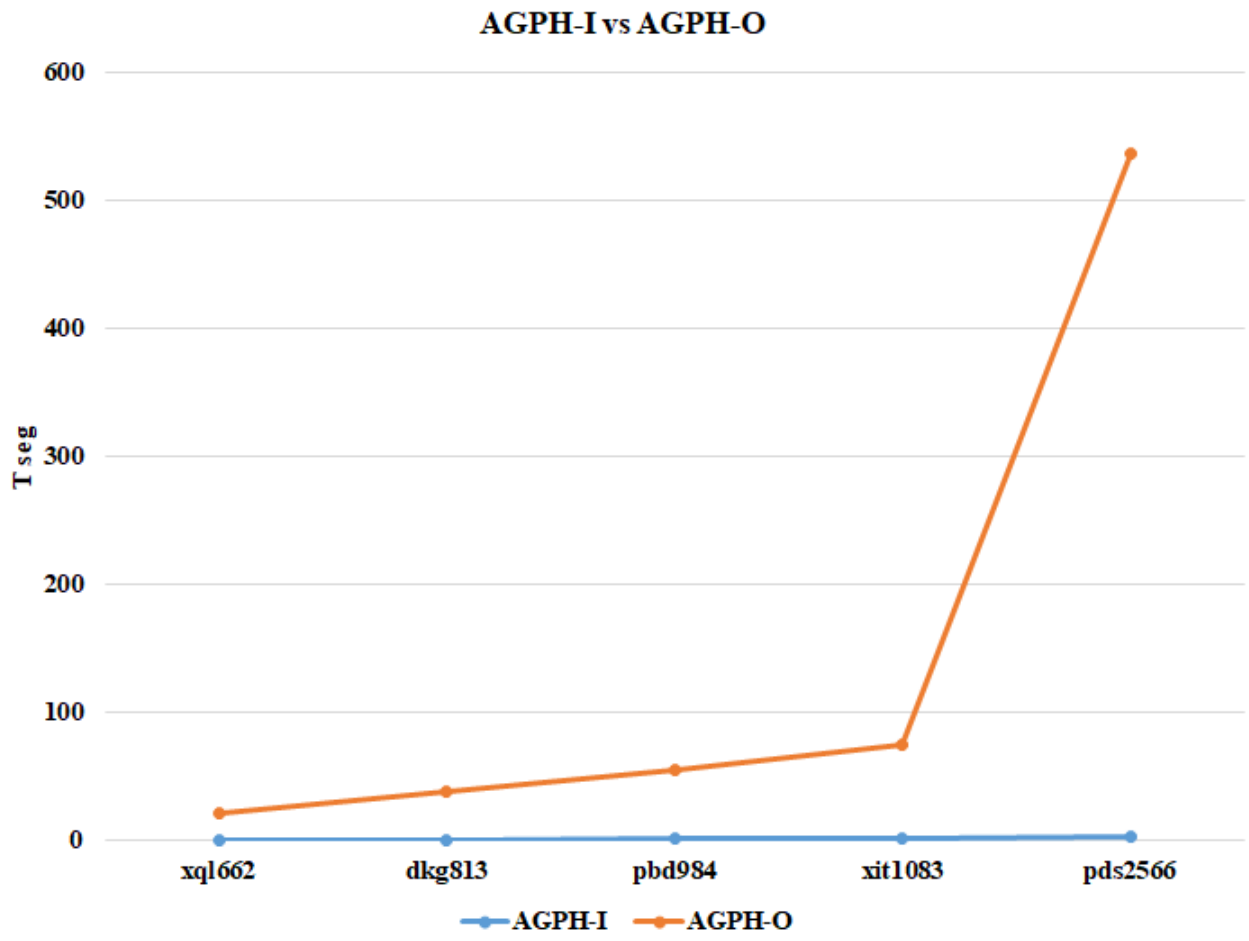


Figura 5.1: Diferencias en tiempo de ejecución de los algoritmos AGPH-I y AGPH-O.

5.4. Instancias del TSP con menos de 1000 ciudades

Para instancias con menos de 1000 ciudades nuestra implementación logra obtener la solución óptima, en tiempos relativamente pequeños. En los resultados obtenidos se destaca que el error calculado es de 0.00 %, este error se presenta para todas las instancias de tamaño menor a 1000 ciudades.

En la Tabla 5.2 se muestran resultados para las instancias xql662.tsp y dkg813.tsp.

Tabla 5.2: Resultados de instancias de tamaño menor a 1,000 ciudades.

PROB	POB	ISLAS	GEN	OLIB	Min	P	E %	T seg
XQL662	32	8	240	2,513	2,513	2,563	0.00	28 seg
DKG813	32	16	1,382	3,199	3,199	3,218	0.00	613 seg

En la Fig. 5.2 se despliega la solución de la ruta óptima obtenida para xql662.tsp, se resalta que el valor del óptimo encontrado y el valor del óptimo reportado en la biblioteca TSPLIB es el mismo; sin embargo, las rutas óptimas son diferentes.

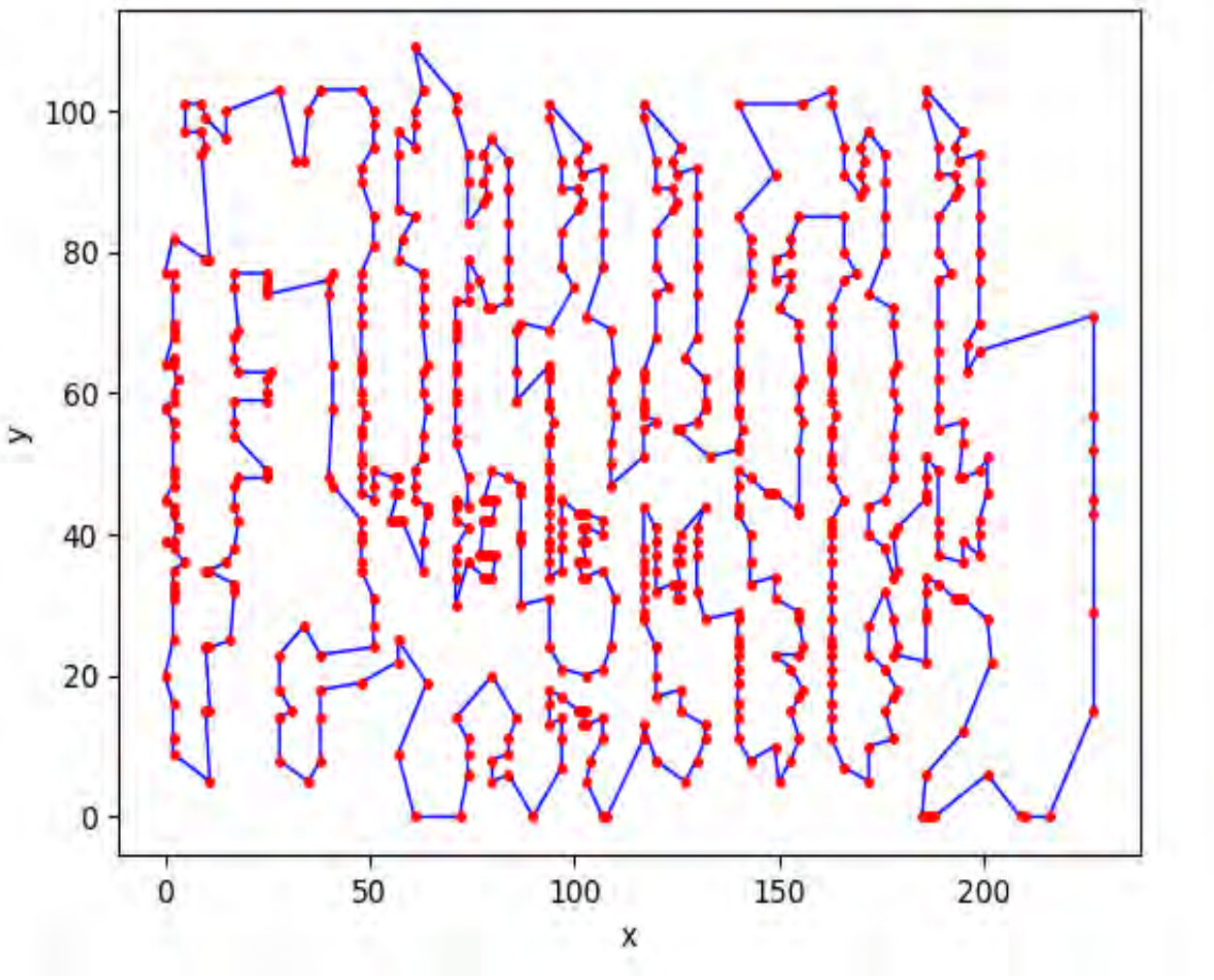


Figura 5.2: Ruta óptima para la instancia xql662.tsp.

5.5. Instancias del TSP entre 1000 y 10000 ciudades

En la Tabla 5.3 se presentan resultados para la instancia xsc6880.tsp y lap7454. En cuanto a la complejidad, estos problemas superan por mucho a los resultados obtenidos para instancias de tamaño menor a 1000 ciudades, sin embargo, el porcentaje de error obtenido sigue siendo

pequeño.

Tabla 5.3: Resultados de instancias de tamaño mayor a 1,000 y menor a 10,000.

PROB	**POB**	**ISLAS**	**GEN**	**OLIB**	**Min**	**P**	**E %**	**T seg**
XSC6880	32	16	1,300	21,535	21,811	21,884	1.28	2,816 seg
LAP7454	40	20	1,500	19,535	19,708	19,742	0.89	6,902 seg

En la Fig. 5.3 se grafica la solución de la ruta óptima obtenida para xsc6880.tsp.

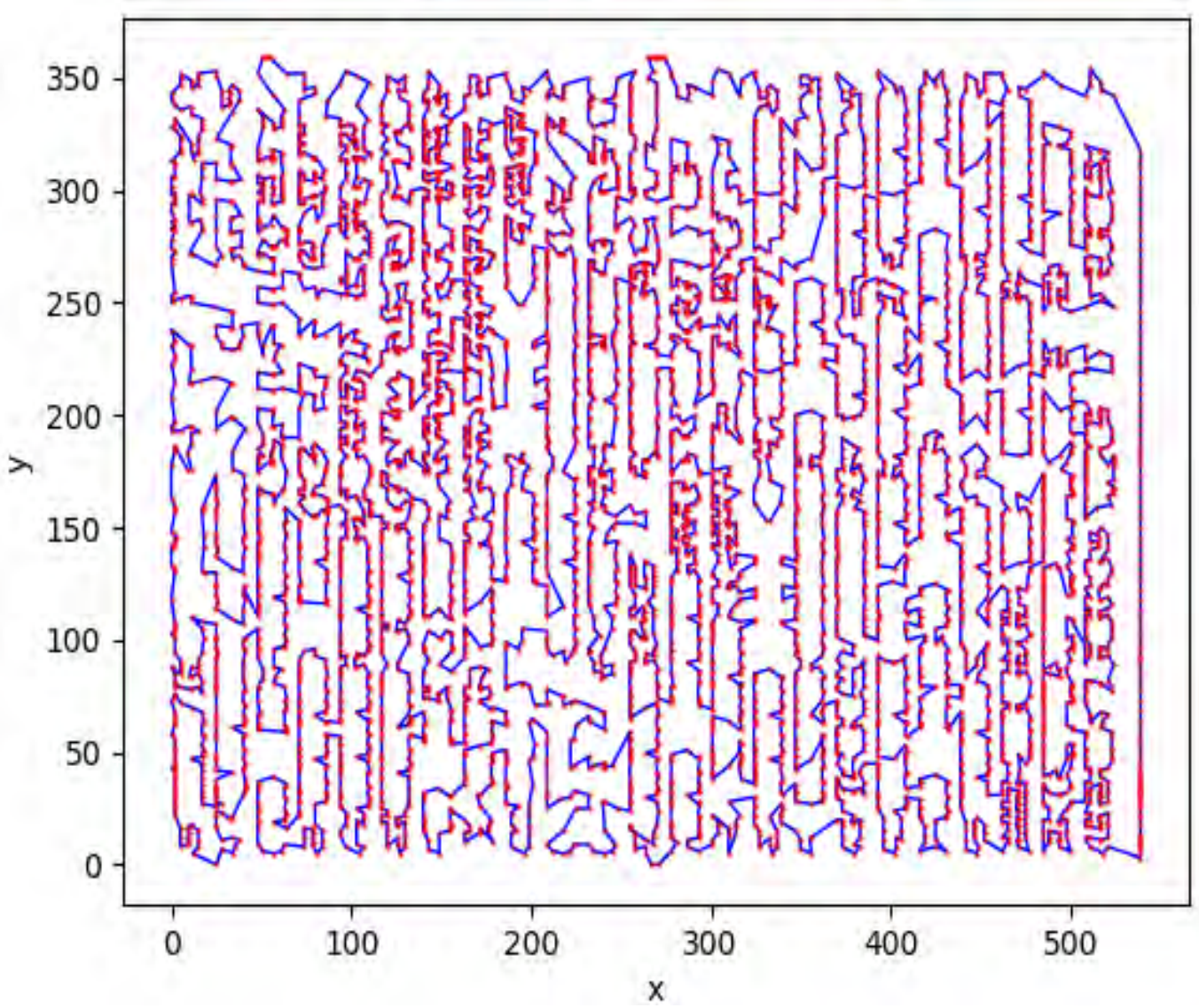


Figura 5.3: Ruta óptima encontrada para la instancia xsc6880.tsp.

5.6. Instancias del TSP con más de 10000 ciudades

Los resultados obtenidos para estas instancias se presentan en la tabla 5.4, estos evidencian que es posible obtener buenos resultados para instancias del orden de decenas de miles de ciudades en tiempos de ejecución razonables, utilizando la biblioteca de funciones estándar POSIX para C en procesadores multinúcleos con suficiente capacidad de memoria.

Tabla 5.4: Resultados de instancias de tamaño mayor a 10,000 y menor a 50,000.

PROB	**POB**	**ISLAS**	**GEN**	**OLIB**	**Min**	**P**	**E %**	**T seg**
XMC10150	60	20	1,500	28,387	28,690	28,903	1.07	2,544 seg
XVB13584	32	8	2,200	37,083	37,751	37,830	1.80	2,476 seg
IDO21215	32	16	2,700	63,517	64,840	65,000	2.08	4,479 seg
PBH30440	32	16	2,700	88,327	90,568	90,699	2.64	6,073 seg
RBZ43748	40	20	4,300	125,183	128,577	128,689	2.71	20,233 seg

En la Fig. 5.4 se grafica la solución de la ruta óptima obtenida para xsc6880.tsp.

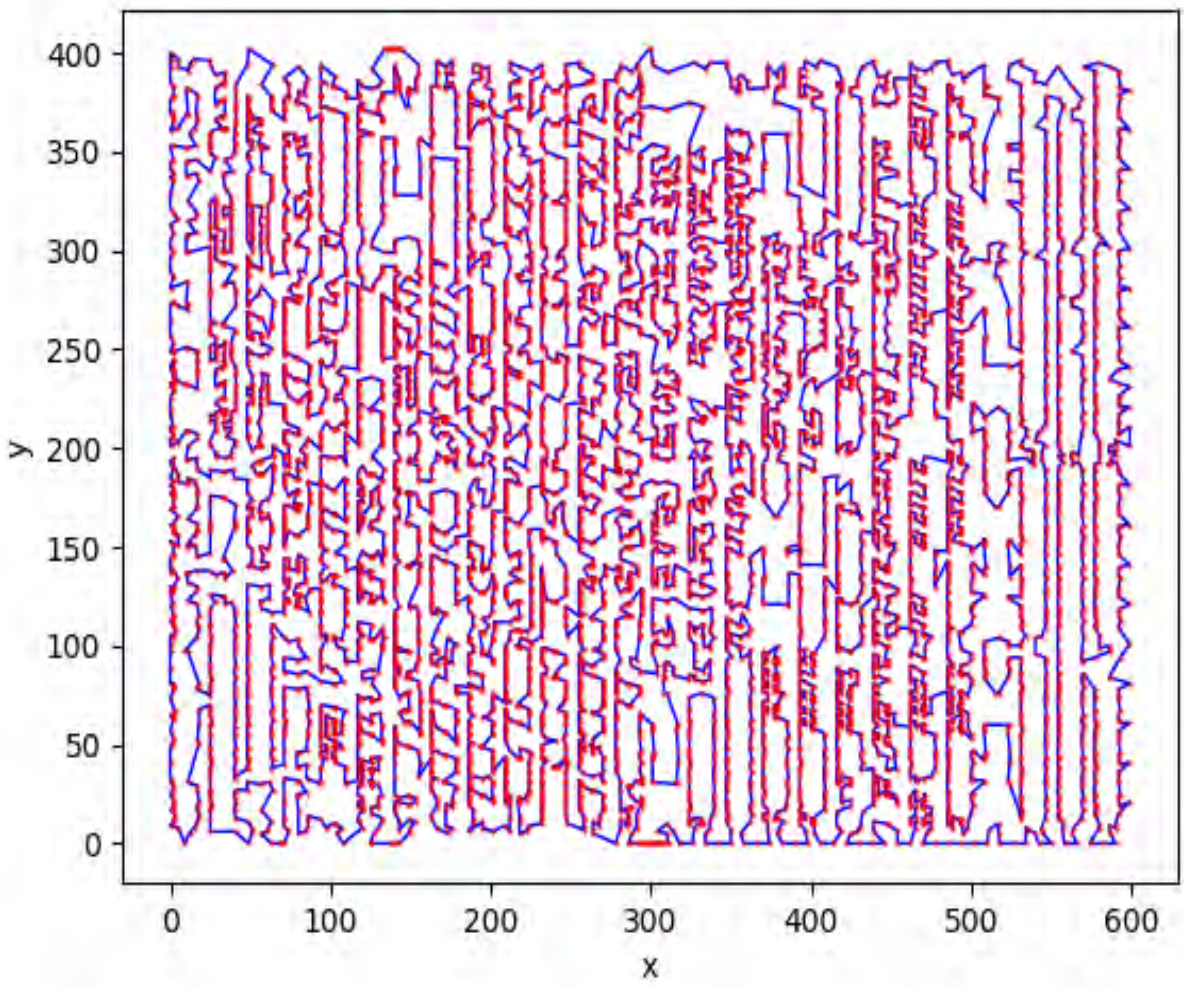


Figura 5.4: Ruta óptima para la instancia xmc10150.tsp.

La Fig. 5.5 muestra de forma gráfica el recorrido generado por la solución obtenida para la instancia rbz43748. Debido a la cantidad de nodos del problema es difícil inspeccionar detalladamente este recorrido, sin embargo, se puede apreciar a simple vista que no contiene cruces de aristas demasiado grandes. El porcentaje de error es considerablemente pequeño dada la cantidad de nodos de esta instancia.

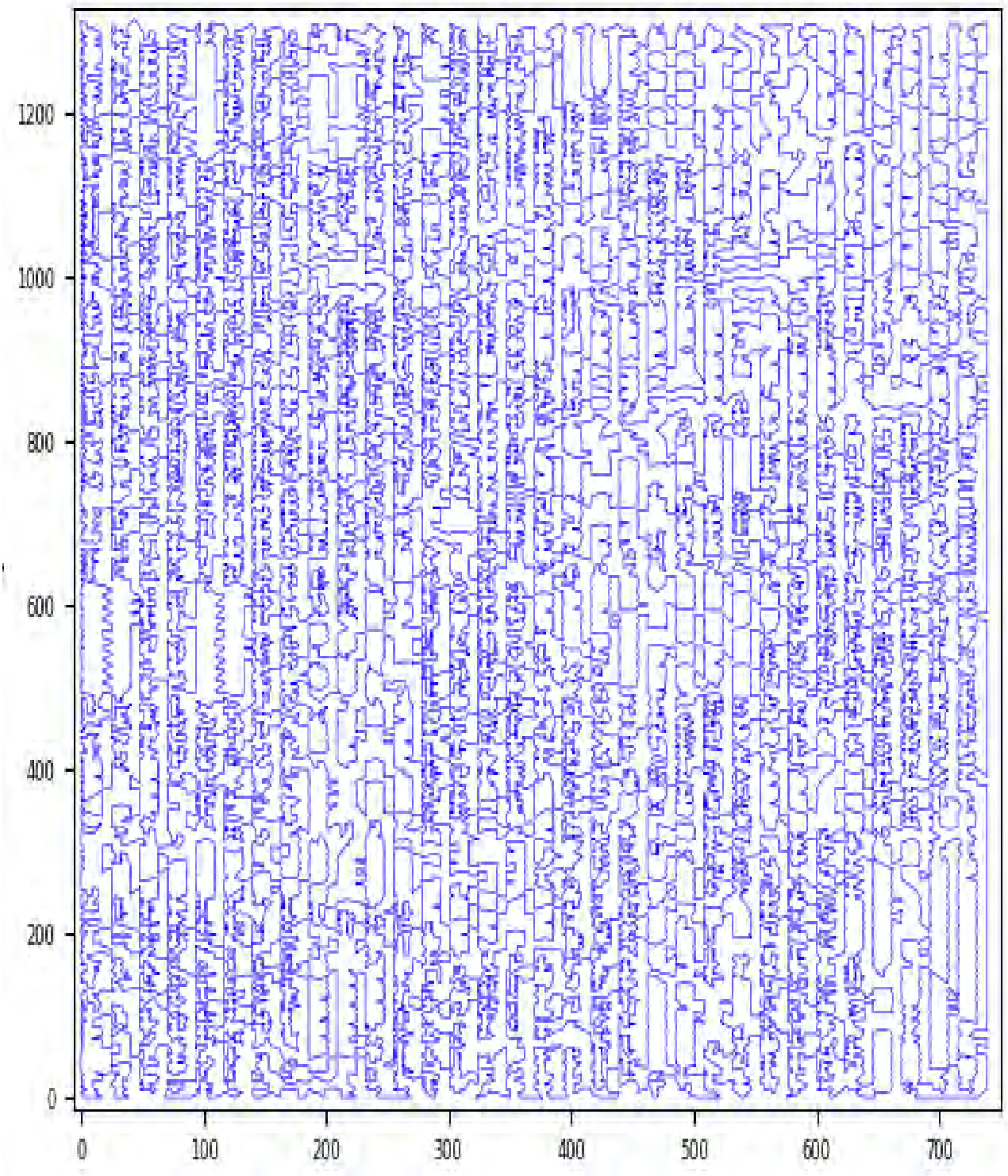


Figura 5.5: Ruta óptima para la instancia rbz43748.tsp.

5.7. Comparación de resultados

Se implementaron tres versiones del algoritmo genético híbrido, el primero es una versión en paralelo con un modelo de islas (AGPH-I), la segunda implementación secuencial con la cantidad de ciudades igual a la de una isla, la tercera implementación es secuencial con el tamaño de la población igual al total de las islas. Los resultados obtenidos se muestran en las Tablas 5.5, 5.6 y 5.7.

La población de individuos en cada isla es de 32, la cantidad de islas es 16, la profundidad para el vecino más cercano es de 40. Se realizaron 10 corridas y se determinó el mínimo, máximo y promedio. Se observa que en todos los casos el porcentaje de error E es menor para el algoritmo AGPH-I.

Tabla 5.5: Resultados del algoritmo AGPH-I.

PROB	**OLIB**	**GEN**	**Min**	**Max**	**P**	**E %**	**T seg**
lap7454	19535	1000	19772	19841	19813	**1.21**	272
ida8197	22338	1000	22692	22747	22718	**1.58**	305
dga9698	27724	1200	28160	28266	28208	**1.57**	816
xrb14233	45450	1500	46368	46479	46427	**2.02**	662

Tabla 5.6: Resultados del algoritmo AGH-S.

PROB	**OLIB**	**GEN**	**Min**	**Max**	**P**	**E %**	**T seg**
lap7454	19535	1000	19834	19933	19887	1.53	311
ida8197	22338	1000	22719	22831	22796	1.71	325
dga9698	27724	1200	28253	28448	28361	1.91	594
xrb14233	45450	1500	46501	46688	46605	2.31	614

Tabla 5.7: Resultados del algoritmo AGPH-P.

PROB	**OLIB**	**GEN**	**Min**	**Max**	**P**	**E %**	**T seg**
lap7454	19535	1000	19844	19886	19866	1.58	2401
ida8197	22338	1000	22714	22807	22766	1.68	2757
dga9698	27724	1200	28274	28284	28273	1.98	3075
xrb14233	45450	1500	46428	46542	46500	2.15	7914

Los errores de las Tablas 5.5, 5.6 y 5.7 se muestran de manera resumida en la Fig. 5.6, se observa que el error es menor con el algoritmo AGPH-I.

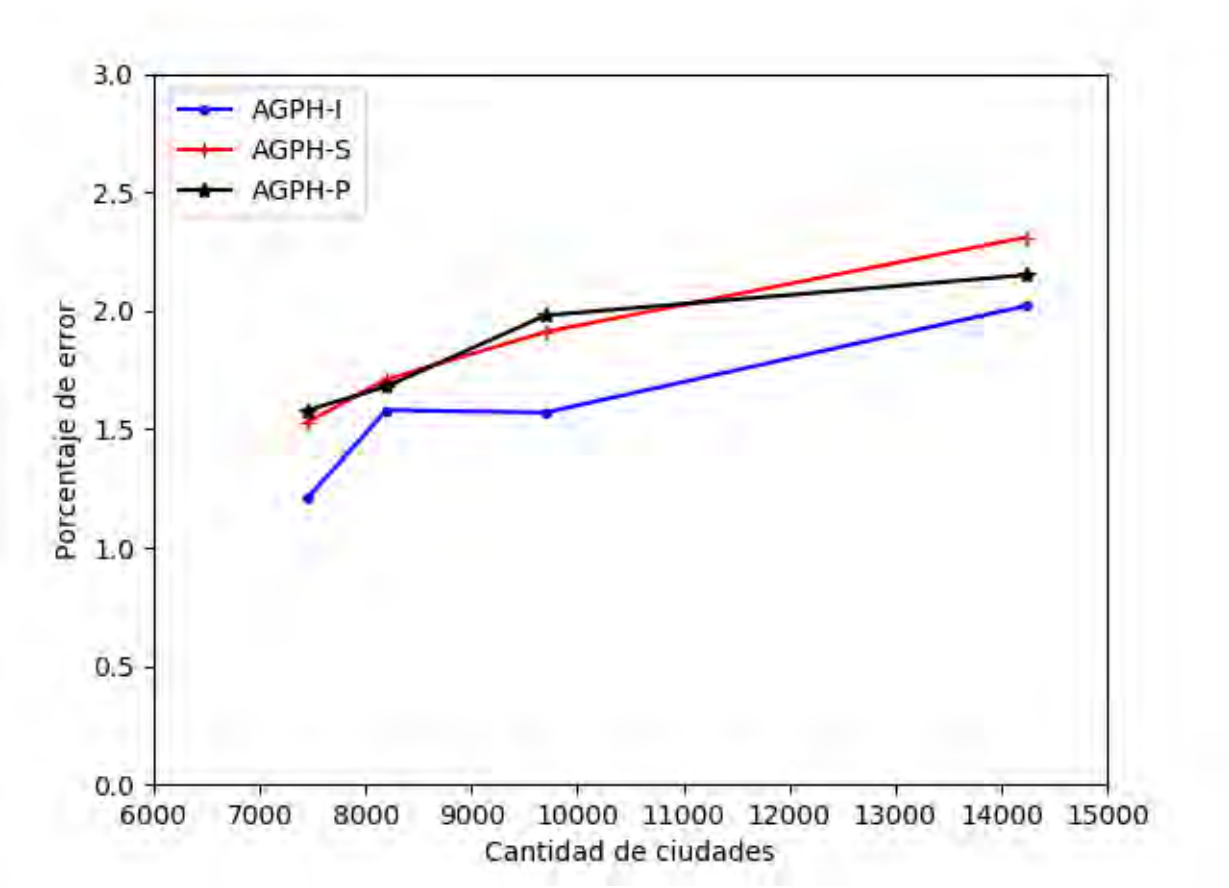


Figura 5.6: Comparación del error para los algoritmos AGPH-I, AGH-S y AGPH-P.

5.8. Discusión

En este trabajo se presenta la implementación de un algoritmo genético paralelo híbrido con modelo de islas (AGPH-I), la paralelización se realizó mediante programación en memoria compartida con hilos POSIX. Se aplicó un modelo de islas con un mecanismo de migración de individuos basado en la topología de anillo. Se observó que si se hace migración desde un inicio todas las islas caen rápidamente en el mismo óptimo local. Por lo que se definió un parámetro que indica la cantidad de generaciones que se deben llevar a cabo antes de empezar a hacer migración. Además, se definió otro parámetro para indicar la cantidad de generaciones entre cada migración.

Para la generación de la población inicial se implementó un algoritmo de búsqueda local muy eficiente, el error en la población es del orden de un 10 % para problemas con menos de 40000 ciudades. Para la instancia rbz43748.tsp en tan solo 10 min llega al 11.51 % de error para

el mejor individuo de la población inicial.

Para la evolución de la población de cada isla se aplicó el operador genético EAX representado por un grafo doble. Este operador genético se combina con búsqueda local lo que hace robusto al algoritmo AGPH-I.

En el algoritmo AGPH-I las primeras generaciones se calculan en un tiempo pequeño, a medida que se avanza el tiempo de ejecución para cada generación se va haciendo más grande, de tal manera que cuando se tiene un porcentaje de error pequeño se requiere mucho cómputo para disminuir el error.

Capítulo 6

Conclusiones y trabajo futuro

Se implementó un algoritmo algoritmo genético paralelo con modelo de islas en combinación con un algoritmo de búsqueda local y el operador de cruce EAX. De esta manera se obtuvo un algoritmo genético híbrido que se implementó en una arquitectura de memoria compartida mediante el uso de hilos POSIX. La paralelización se llevó a cabo mediante el modelo de islas con migración de individuos en una topología de anillo. La migración de los individuos entre las islas contribuyó a tener diversidad de los mejores individuos en las islas y aceleró la convergencia.

El algoritmo se aplicó para resolver el problema del agente viajero con instancias de decenas de miles de ciudades, con un error menor al 2 %.

6.1. Conclusiones

Las principales conclusiones derivadas de los resultados de esta tesis se enlistan a continuación:

- Una buena implementación del algoritmo de búsqueda local permite disminuir significativamente el tiempo de ejecución para encontrar la población inicial.
- El modelo de islas reduce el tiempo de ejecución al migrar individuos entre las islas y además se tiene un porcentaje de error menor.
- El operador EAX combinado con búsqueda local 2-opt permite encontrar soluciones muy cercanas al óptimo.

6.2. Trabajo futuro

La investigación desarrollada en esta tesis se puede continuar desde varios enfoques, desde el punto de vista de la paralelización, de las topologías para la migración de individuos, el análisis de los parámetros. Para el trabajo futuro se plantea lo siguiente:

- En el algoritmo AGPH-I implementado los valores de los parámetros son estáticos, es decir mantienen su valor durante toda la ejecución. El algoritmo AGPH-I podría mejorar si se implementan parámetros dinámicos, es decir que al inicio de la ejecución los parámetros tengan un valor y a medida que el algoritmo se ejecute los valores de los parámetros se ajusten automáticamente, mediante alguna estrategia de optimización.
- Las ideas en el desarrollo del algoritmo AGPH-I se pueden extender a la implantación en una arquitectura con memoria distribuida en un clúster de computadoras.
- El algoritmo AGPH-I también se puede implementar en arquitecturas de procesadores gráficos.

Referencias

[1] J. V. de Oliveira, S. Baltazar, and H. Daniel, "On asynchronous parallelization of order-based GA over grid-enabled heterogenous commodity hardware," *Soft Computing*, vol. 21, no. 21, pp. 6351–6368, 2017.

[2] J. Robinson, S. Vrbsky, X. Hong, and B. Eddy, "Analysis of a high-performance TSP solver on the GPU," *Journal of Experimental Algorithmics (JEA)*, vol. 23, no. 1, pp. 1–22, 2018.

[3] Y. Wu, T. Weise, and R. Chiong, "Local search for the traveling salesman problem: A comparative study," in *2015 IEEE 14th International Conference on Cognitive Informatics Cognitive Computing (ICCI*CC)*, pp. 213–220, 2015.

[4] Z. Mu, J. Dubois-Lacoste, H. H. Hoos, and T. Stützle, "On the empirical scaling of running time for finding optimal solutions to the TSP," *Journal of Heuristics*, vol. 24, no. 6, pp. 879–898, 2018.

[5] Y. Nagata and S. Kobayashi, "A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem," *INFORMS Journal on Computing*, vol. 25, no. 2, pp. 346–363, 2013.

[6] K. Honda, Y. Nagata, and I. Ono, "A parallel genetic algorithm with edge assembly crossover for 100,000-city scale TSPs," pp. 1278–1285, 2013.

[7] M. M. Alipour, M. M. Alipour, S. N. Razavi, S. N. Razavi, M. R. Feizi Derakhshi, M. R. Feizi Derakhshi, M. A. Balafar, and M. A. Balafar, "A hybrid algorithm using a genetic algorithm and multiagent reinforcement learning heuristic to solve the traveling salesman problem," *Neural Computing and Applications*, vol. 30, no. 9, pp. 2935–2951, 2018.

[8] D. Davendra, *Traveling Salesman Problem, Theory and Applications*. IntechOpen, 2010.

[9] G. Gutin and A. P. Punnen, *The Traveling Salesman Problen and Its Variations*. Springer, 2002.

[10] B. Reus, *Limits of Computation: From a Programming Perspective*. Cham: Springer, 2016.

[11] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms, Second Edition*. McGraw-Hill Higher Education, 2nd ed., 2001.

[12] D. Levy and L. Pachter, "The neighbor-net algorithm," *Advances in Applied Mathematics*, vol. 47, no. 2, pp. 240–258, 2011.

[13] K. Helsgaun, "An effective implementation of the lin–kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.

[14] "Concorde tsp solver." www.math.uwaterloo.ca/tsp/concorde/index.html.

[15] E. Cantú-Paz, *Eficient and Acurate Parallel Genetic Algorithms*. Kluwer Academic Publisher, 2001.

[16] O. Kramer, *Genetic Algorithm Essentials*, vol. 679. Cham: Springer, 2017.

[17] S. N. Sivanandam, S. N. Deepa, and S. O. service), *Introduction to Genetic Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[18] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *CoRR*, vol. abs/1109.3627, 2011.

[19] W. Mendenhall, R. J. Beaver, and B. M. Beaver, *Introducción a la probabilidad y estadística*. Cengage Learning, 2010.

[20] C. A. Coello, *Introducción a la Computación Evolutiva (Notas del Curso)*. CINVESTAV-IPN, 2020.

[21] J. R. Cheng and M. Gen, "Accelerating genetic algorithms with GPU computing: A selective overview," *Computers and Industrial Engineering*, vol. 128, pp. 514–525, 2019.

[22] T. Rauber and G. Rünger, *Parallel Programming: For Multicore and Cluster Systems*. Berlin/Heidelberg: Springer, 2010.

[23] Ko-cha, "Edge-assembly-crossover." www.github.com/ko-cha/Edge-Assembly-Crossover, Nov 2018.

[24] U. Heidelberg, "Solving TSPs." www.math.uwaterloo.ca/tsp/vlsi/index.html, 2013.

Apéndice A

```
Instancia TSP/xql662.tsp
Cantidad de hilos = 8
Cantidad generaciones antes de migración = 200
Cantidad iteraciones entre migracion = 40
Cantidad veces que se hará migración = 20
Tamaño de la poblacion = 32
Cantidad de descendientes = 32
Profundidaad busqueda local = 50
Hilo   gen   min   max   Prom.   error
0      0  2719  2870  2795.03    8.20    0 seg
1      0  2702  2853  2792.25    7.52
2      0  2738  2874  2801.31    8.95
3      0  2720  2851  2797.47    8.24
4      0  2714  2872  2796.03    8.00
5      0  2722  2885  2799.94    8.32
6      0  2734  2862  2800.47    8.79
7      0  2722  2878  2791.22    8.32
0    200  2515  2564  2539.53    0.08    23 seg
1    200  2519  2576  2543.03    0.24
2    200  2526  2562  2541.34    0.52
3    200  2523  2567  2544.31    0.40
4    200  2520  2567  2543.31    0.28
5    200  2514  2568  2544.25    0.04
6    200  2534  2563  2545.06    0.84
7    200  2523  2568  2542.66    0.40
0    240  2515  2557  2537.06    0.08    28 seg
1    240  2519  2563  2540.69    0.24
2    240  2523  2558  2539.16    0.40
3    240  2520  2566  2541.62    0.28
4    240  2513  2563  2540.31    0.00
Archivo de salida xql662_771.tour
Tiempo 28 seg
```

Bla bla bla

```
NAME: dkg813_288.tour
```

```
COMMENT: Tour length  3199
TYPE: TOUR
DIMENSION: 813
TOUR_SECTION
122 121 230 231 248 260 247 259 246 302 303 304 305 277 251 262
250 261 249 232 233 234 185 235 236 237 238 278 254 264 253 263
252 321 306 307 308 309 397 413 416 446 449 450 445 444 485 490
512 511 524 510 509 508 507 506 505 484 464 483 463 455 437 438
439 440 441 442 443 396 395 394 393 392 391 390 389 388 387 436
435 434 385 479 480 481 482 504 503 502 522 633 635 636 637 638
700 639 640 758 802 796 795 742 794 793 792 791 741 790 789 740
788 810 811 787 786 739 785 784 783 782 737 781 780 779 778 736
735 734 777 776 809 774 775 733 732 708 694 666 651 695 667 652
668 696 709 669 697 710 698 670 653 632 631 630 576 575 574 628
627 573 572 626 625 692 665 693 748 731 730 707 664 691 663 690
726 727 728 729 808 725 724 689 650 647 649 662 688 723 757 722
687 686 614 613 648 661 685 683 721 756 773 807 806 805 804 772
755 720 754 682 719 718 681 660 680 717 716 715 706 659 678 714
803 713 712 711 705 658 753 770 752 751 769 750 657 749 768 767
766 799 765 677 764 798 763 797 762 761 759 747 704 676 675 703
746 813 812 801 800 760 771 679 684 612 617 629 634 577 578 641
672 701 671 699 738 743 744 745 702 656 674 655 673 654 642 643
579 580 581 644 645 582 583 584 586 587 588 589 601 590 591 592
593 594 536 528 537 538 595 596 602 597 598 599 539 540 541 515
517 529 542 543 600 544 545 546 547 548 550 603 646 604 605 551
606 607 608 609 552 610 611 553 554 555 556 557 558 559 560 561
562 563 564 615 616 565 566 567 568 618 619 620 569 621 622 623
624 571 570 532 500 476 477 533 501 478 462 433 384 383 382 432
431 489 499 498 497 475 430 429 428 379 378 377 427 376 375 426
374 373 425 372 371 370 424 423 369 368 367 366 422 454 461 474
496 495 494 493 492 473 460 472 459 417 418 419 420 421 365 364
363 362 360 358 458 471 470 457 451 453 456 469 491 520 519 518
468 452 350 351 352 353 354 355 356 357 317 318 289 283 240 210
181 160 180 179 178 177 145 144 143 159 142 141 158 140 157 139
138  86 137 156 167 168 169 170 171 172 173 174 175 176 267 334
316 315 333 314 332 313 312 311 266 256 310 347 348 346 344 415
```

410 409 408 343 331 342 330 407 406 341 329 328 340 327 339 326
325 282 324 338 337 323 322 336 403 404 405 467 488 527 535 526
534 525 513 486 466 465 523 531 521 530 549 585 516 514 487 411
448 345 349 412 359 361 380 381 386 447 398 399 400 414 401 402
335 279 320 294 291 271 268 290 288 287 265 281 239 208 209 211
216 221 184 113 108 107 132 134 75 79 73 44 46 80 83 81
84 95 82 57 27 30 68 69 70 71 72 99 131 130 106 98
97 128 129 186 255 280 187 188 189 190 191 192 193 194 195 196
197 198 199 200 201 202 207 203 204 205 206 166 136 155 135 165
154 153 101 85 100 133 164 152 163 151 150 149 162 148 161 31
32 33 34 35 36 37 43 5 38 39 74 40 41 42 6 7
8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 1 2 25 45 47 102 109 88 87 146 183 213 212 182 241
269 270 319 272 214 215 273 284 217 218 219 220 274 292 293 295
296 297 285 245 258 244 257 243 222 223 224 225 226 242 298 299
300 275 286 301 276 229 228 227 147 120 93 119 118 117 116 92
115 114 91 90 112 111 89 110 103 48 49 50 51 52 53 3
54 55 56 76 58 59 60 26 94 104 61 62 77 63 64 78
65 66 4 29 28 67 96 127 105 126 125 124 123